SESSION

SOFTWARE MODELING, DESIGN METHODOLOGIES AND MODEL DRIVEN ENGINEERING

Chair(s)

TBA

Application of a Communication Protocol Methodology to Embed a Collaborative Virtual Reality Environment in Building Industry Game

Lacey Duckworth, Dr. Tulio Sulbaran, Dr. Andrew Strelzoff, Professor Hal Johnston

***Abstract*— The Building Industry Game (B.I.G.) is a web based simulation that allows Construction Management students to take the role of contractors and make bids on various jobs using their own in-game finances. In order to win a bid, students need to make the lowest bid and adequate cash-on-hand. The bids made by students are on particular job types that exist in the construction field including: apartment, hospital, industrial, office, and school. For each given job type, students also have to select a method they wish to use for each of the following activities: excavation, foundation, basement, framing, closure, roof, siding, finishing, and mechanical/electrical. Each of these methods have a certain labor and material costs where the higher the value implies more workers and equipment or bigger equipment. When selecting a method for a particular job type, students rely on numerical values to help determine what bid they should make. Students would benefit from a visual to show the materials and labor cost associated with each method in order to select the best method and make the winning bid. This paper discusses the use of a communication protocol developed by Lacey Duckworth to embed a Collaborative Virtual Reality Environment, Open Source Building Environment for Simulation and Training (OSBEST), into B.I.G. in order to provide a visual for the materials and labor costs of a particular job type and excavation method selected by the user.**

***Index Terms*— Simulation, Collaborative Virtual Reality Environment, Communication Protocol, Methodology, Game**

I. Introduction

SIMULAITONS provide the means to visualize the characteristics of data while minimizing the risks associated with decision making and problem solving. The Building Industry Game (B.I.G.) is a simulation that allows students in Construction Management to take the role of a contractor and make bids on jobs based on the job size, type, and method for an activity associated with the job. B.I.G is a text, web-based simulation and does not have a 3-dimensional visual of the materials, equipment, or workers involved in each of the particular methods. One way to incorporate a 3-dimensional visual in B.I.G is to embed a collaborative virtual reality environment (CVRE) in the simulation so that students can select a method for a particular job and see the materials, equipment and workers involved in for the particular job type and size. This paper discusses the use of a communication protocol methodology developed by Lacey Duckworth to embed a CVRE, Open Source Building Environment for Simulation and Training (OSBEST), into B.I.G.

Lacey Duckworth is with The University of Southern Mississippi, School of Computing, 118 College Drive Box 5106, (corresponding author phone: 601-266-4895; fax: 601.266-5792; e-mail: Lacey.Duckworth@eagles.usm.edu).

Dr. Tulio Sulbaran is with The University of Southern Mississippi, School of Construction, 118 College Drive Box 5138 (email: Tulio.Sulbaran@usm.edu).

Dr. Andrew Strelzoff is with The University of Southern Mississippi, School of Computing, 118 College Drive Box 5106 (email: Andrew.Strelzoff@usm.edu).

Professor Hal Johnston is with California Polytechnic State University, Construction Management Department, 1 Grand Avenue, San Luis Obispo, CA93407 (email: hjohnsto@calpoly.edu).

II. Building Industry Game

The Building Industry Game (B.I.G.) was envisioned and created by Glenn Sears in from the University of New Mexico. In the late 1980's Jim Borland, from California Polytechnic State University, was given permission from Sears to use and rewrite the game. Using Sears' and Borland's ideas, B.I.G. was transformed into a web-based game written using JAVA and POSTGRES by the California Polytechnic State University Multimedia Group with the help and direction of Borland and Professor Hal Johnston. B.I.G. was designed as a text, web-based simulation of a construction business where Construction Management students, age 25 and under, assume the role of a contractor making bids on various jobs with the goal of winning the bid. Each student manages their own company and as in the real-world responsible for maintaining their financial information such as balance sheets, income statements, and schedules. In order to win a bid, each student needs to ensure that they have cash-on-hand to make cover the bid as well as make the lowest bid amongst their competitors. In order to help determine what bid they should make, each student analyzes the type and size of job as well as the method for activities to complete a job [1].

In B.I.G. there are five different job types for which a student can make a bid. These jobs include a(n) school, apartments, office, hospital, or industrial plants. Each of these particular jobs are given a randomized job size of a particular range set by the admin before the game begins. Figure 1 shows an example some job types and the random generated job size. Students refer to the job type to determine the shape of excavation to be made and the job size to determine what materials and labor costs are associated with a given job [1].

Job Type	Job Size (CUs)
Public Apartments	20937
Public Industrial	43132
Public Industrial	45983
Public School	31974
Public Office	35526
Public Apartments	22191
Public Apartments	23182
Public School	37291
Public Apartments	17900

Fig. 1. Example of Job Type and Job Size in a game instance of B.I.G.

Using the job type and size, students select a method to use for each activity. The activities associated with each job include excavation, foundation, basement, framing, closure, roof, siding, finishing, and mechanical/electrical. For each of these activities, there is a particular method that has a given labor cost, material cost, and number of days required to complete the project. By selecting the best method and making the best bid based on the methods selected for each activity, the student can win a bid assuming they have the cash-on-hand to cover the bid. Like the job size, the different costs assigned to a particular method are a randomized value from a particular range specified by the game admin. For each given activity, there are five different methods [1]. Figure 2 shows two of the different methods for each activity.

	Method #1				Method #2		
	Labor ($)	Material ($)	Subcontractor ($)	Days	Labor ($)	Material ($)	Subcontractor ($)
Excavation	25333	23030	0	35	41454	27636	0
Foundation	34545	27636	0	41	62181	34545	0
Basement	20727	25333	0	47	69090	36848	0
Framing	50260	40208	0	62	92981	52773	0
Closure	80605	23030	0	57	85211	80605	0
Roof	50666	50666	0	71	82908	69090	0
Siding	69090	57575	0	71	145089	75999	0
Finishing	80605	92120	0	95	115150	184240	0
Mech/Elec	0	0	149695	143	0	0	207270

Fig. 2 Two methods for each job type.

When selecting a job to bid on, students refer to the size of the job and method they wish to use in order to determine what bid they should make on a particular job. Figure 3 shows the GUI that allows students to select the method they wish to use for each activity and make their bid. In addition, to making bids, students also have the ability to retract a bid that was previously made or cancel out of making an initial bid. When the student selects a method to use for an activity they would benefit from seeing what the labor and material costs are associated with. In other words, it would be beneficial to for the students to see how many workers, what type of equipment and how many of each is accumulated in the labor and material costs. The next section talks about how Open Source Building Environment for Simulation and Training (OSBEST) can be used to provide a visual for this part of the game.

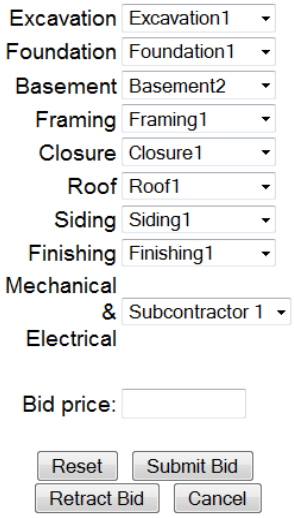


Fig. 3 GUI for selecting methods and placing a bid.

III. Open Source Building Environment for Simulation and Training (OSBEST)

Open Source Building Environment for Simulation and Training (OSBEST) is an open source virtual reality environment platform BSD licensed by The University of Southern Mississippi. It is currently the first Web-GL (a client virtual reality environment) based JavaScript virtual environment although, in the past, it has been based on various other client virtual reality environments including SecondLife, realXtend, and O3D.

The purpose of OSBEST is to enhance current practices in Architectural Engineering and Construction through the application of technology, in this case through the use of virtual reality environments. OSBEST, in its current and previous forms, allows users from around the world to log into a virtual environment and interact with various individuals in order to complete a particular task. Once the capabilities of OSBEST were recognized, it was determined that embedding it into B.I.G. would be possible since since both were web-based, implemented with JAVA. In addition to the implementation simularities, OSBEST was also considered because previous work had been done in the virtual environment which included various construction operations including excavating, loading and placement of objects, and various other elements which are used to create virtual environments.

IV. Embedding OSBEST into B.I.G

Instead of using an ad-hoc design to embed OSBEST into B.I.G., a communication protocol methodology developed by Lacey Duckworth at The University of Southern Mississippi was used. This methodology would make embedding OSBEST in B.I.G. easy and in the future if another method was to be added, the communication protocol could quickly be modified to include this new method.

A. Communication Protocol Methodology

The communication protocol methodology is a set of steps which utilize a state diagram and transition table, and a communication protocol diagram and components table which allows programmers to design a project that can be easily implemented in order to prevent "spaghetti-code" by providing easy scalability and increase the performance of the virtual environment as the calculations for the simulation are external to the virtual environment. While methods to address this issue have been studied in the implementation of FlowVR, Solipsis, and the collaborative virtual geographic environment system, an easy to follow methodology has not been provided that will allow non-experts to embed CVRES in Simulations [3, 4, 5].

Steps in the Methodology

The set of steps are as follows:

1) *Understand project at hand*
2) *Determine the languages for the CVRE, Middleware and Simulation language*
3) *Build a test communication between the CVRE and Simulation language selected.*
4) *Develop a state diagram and transition table*
5) *Complete the Communication Protocol Diagram using the specialized components table*
6) *Implement the simulation in the CVRE using the*
7) *Test the simulation in the CVRE*

State Diagram and Transition Table

The state diagram is used to determine calculations need to be performed on the simulation side. There is no new research as far as how this step is implemented. In other words, this is the standard state machine where the circles represent a possible state to enter, and the lines that connect two states are the different triggers that must occur to move to that next state. Figure 4 shows a simple example of the state machine for opening and closing a door. As with any state machine, a

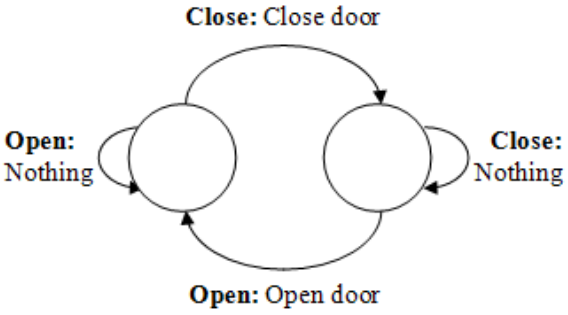


Fig. 4 State machine for simulation of door open and close

transition table exists in order to compare the the various states with the transitions in order to determine what needs to happen. In this instance. The transition table for the close and open door example can be seen in Figure 5. In this methodology, when the various states are compared with the transitions, the intersection of the two gives indication of what needs to happen in the virtual environment. For example, at the intersection of the state, open, and the transition, close, the user has selected to close the door . This means the simulation needs to determine how to shut the door and send this informaiton to the virtual environment to create the visual of the door shutting.

		State	
		Open (0)	**Close (1)**
Transition	**Open (0)**	"Nothing"	"open door"
	Close (1)	"close door"	"Nothing"

Fig. 5 State machine for simulation of door open and close

Communication Protocol Diagram and Components Table

The communication protocol diagram is one similar to the design of a client server. The communication protocol diagram models the communication needs between a client, the virtual environment, and a server, the simulation language. In order to communicate with each other the collaborative virtual reality environment (client) sends messages through some middleware (a communication layer) in order to request information from the simulation language (server). The simulation language (server) keeps up with the information as highlighted in the state machines intersection of states and transitions, interprets the message received from the collaborative virtual reality environment (client), determines what the next state is, and makes the necessary calculations to send back to the collaborative virtual reality environment (client). When the collaborative virtual reality environment receives this message it implements the instructions received as a visual through the use of slaves (objects in the collaborative virtual environment). This process emphasizes the importance of designing the state diagram and transition table as the terms associated with the state machine and its diagram are highlighted on the simulation language side (server). The described communication protocol can be seen in figure 6 where the numerical values in the figure indicate the various components as listed in that need to be implemented.

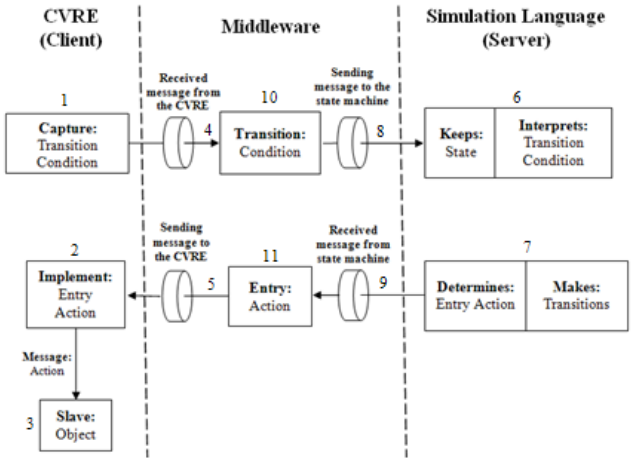


Fig. 4 Communication Protocol

TABLE 1
SPECIALIZED COMPONENTS OF THE COMMUNICAITON PROTOCOL DIAGRAM

CVRE	Simulation Language	Middleware
• (1) CVRE Event Triggers • (2) Primary Event Adapter • (3) Secondary Event Adapter • Adapters: - (4) CVRE to Network - (5) Network to CVRE	• Server-Side State Machine - (6) State keeper and next state determination - (7) Action and transition update • Adapters - (8) Simulation to Network - (9) Network to Simulation	• (10) Condition Transfer • (11) Action Transfer

B. Using the Methodology to Embed OSBEST into B.I.G

In order to embed OSBEST into B.I.G., following the communication protocol methodology above, each step was followed.

Step 1: Understand project at hand

In order to complete this step, several meetings were held with Professor Hal Johnston to determine what type of visual he wanted implemented. After several discussions, it was determined that he wanted visual for the selection of a method for the excavation activity. This meant that given a method for the excavation selected by the students and the job size and type a visual would be created so that students could get an idea of the workers and equipment involved in the exercise. For this project, he job size and method selected by the student was used to determine the workers and equipment involved in the excavation. The job size was represented by a range varying from zero to infinity dollars in increments of 20,000 with an upper limit starting at 80,000 to create four ranges. For each of the ranges, a base number of workers and equipment was established. Combining this information with the method selected by the students, where each method meant bigger, better equipment and less workers provided the means for making the visual for the students to see what workers and equipment would be involved with the method they selected..

Step 2: Determine the languages for the CVRE, Middleware and Simulation language

This step is important as it determines what language will be used to implement the simulation, CVRE, and the middleware. Because B.I.G was already implemented using JAVA and POSTGRES and the version of OSBEST at the time of this project was O3D, implemented with JavaScript Object Notation (json), the only determination to be made was the language for the middleware. The language selected to implement the middleware was PHP as it provided means to pass the data from the simulation language to the CVRE in json packets.

Step 3: Build a test communication between the CVRE and Simulation language selected.

In order to determine if messages could be sent from the simulation language, B.I.G., to the CVRE, O3D, code was implemented to determine if a small packet of json data could be sent to B.I.G. from O3D and visa versus using the selected middleware, PHP, form step 2. The test case to be implemented can be seen in figure 5. This test will prevent time and effort in the future as it will be determined early in the design of the project whether or not the three languages are compatible and capable of communicating with each other.

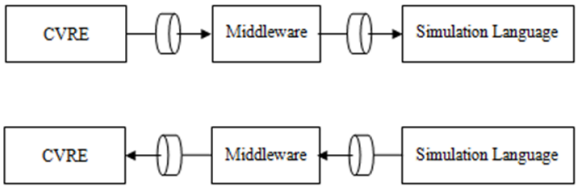


Fig. 5 Example of the communication to be established between the simulation language and the CVRE.

Step 4: Develop a state diagram and transition table

Once the communication between the CVRE and simulation language is complete, the next step is to develop the state diagram, which is the different things (states) that need to occur to visualize the simulation and the different events (transitions) that will require transfers to new states. Figure 6 shows state diagram developed for the simulation of excavation in B.I.G.

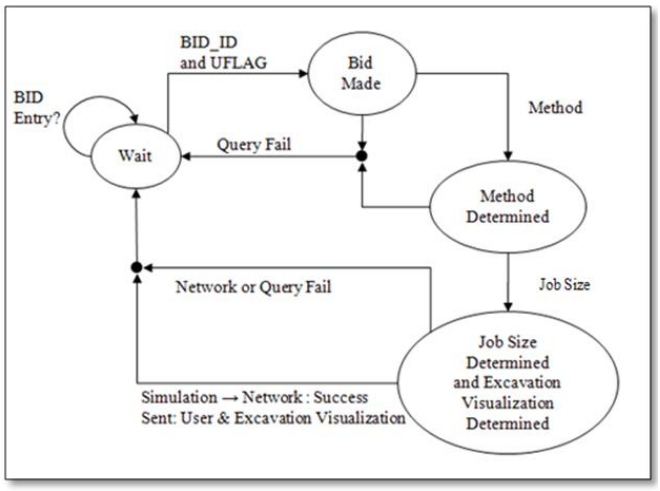


Fig. 6 The state diagram for the order of operations in B.I.G. in order to develop a visual for the simulation.

Using the state diagram, the transition table is designed. In order to design this table, all of the things that need to happen in the simulation (states) and how those events are triggered (transitions) are compared against each other in order to determine what needs to happen in the simulation language in order to move to that next state. Table 2 shows the states compared with the transitions of the events to be modeled in the simulation.

In addition to providing a the order of operations required to implement the simulation this step also provides the calculations that need to occur in the simulation language to prepare the visual in the CVRE through the intersection of each state and transitions. In this example, Once the method had been determined (a particular state), in order to determine what excavation visualization to make (the next state), the job size has to be obtained from the simulation language (the transition). Thus, the simulation language is held responsible for fetching this information and using it to make the visualization to send the CVRE.

TABLE 2
TRANSITION TABLE DESIGNED USING THE STATE DIAGRAM IN FIGURE 6

	Bid DB Entry	**User, BID_ID, UFLAG**	**Meth -od**	**Job Size**	**Sim. -> Network Success Sent: User and Exc. Visual.**
Wait	Query for user, Bid_ID, and user flag. Transition to Bid Made	User has made a bid. Query for the user, Bid_ID, and UFLAG.	-	-	-
Bid Made	-		Query for job size	-	-
Method Determined	-	-	-	Deter-mine the excavation to be made	-
CEU and Excavation Visualiz-ation Determ-ined	-	-	-	-	Feed message to simulation adapter and transition to wait stage

Step 5: Complete the Communication Protocol Diagram using the specialized components table

Using the previous steps, in particular the steps to determine the languages, the communication test, the state diagram and transition table, the communication protocol diagram, figure 4, can be completed while referring to the specialized components, table 1, to provide a guideline for what each component does. Due to the complexity of the communication protocol diagram, the figures 7 - 9 show each component of the communication protocol as outlined in figure four for each individual language. Placing these components in the same orientation as figure 4 will result in the complete communication protocol diagram to make the excavation simulation visual in O3D that was selected by students while interacting with B.I.G.

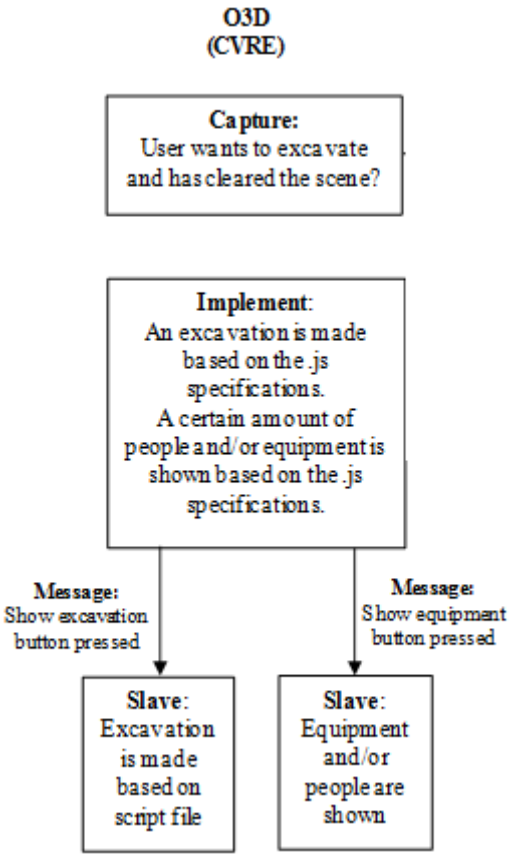


Fig. 7 The specialized components of the communication protocol diagram for the CVRE language, O3D

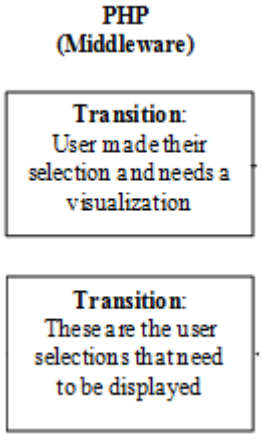


Fig. 8 The specialized components of the communication protocol diagram for the middleware language, PHP

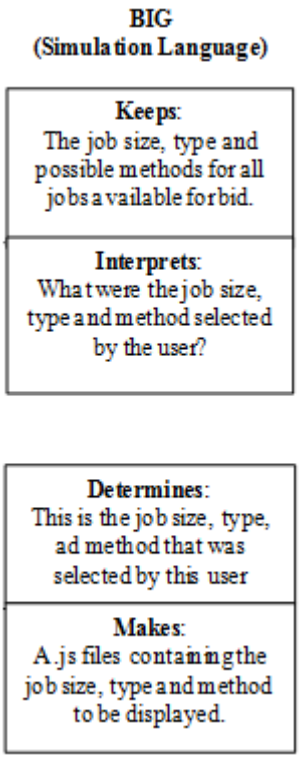


Fig. 9 The specialized components of the communication protocol diagram for the simulation language, B.I.G.

Step 6: Implement the simulation in the CVRE using the

Once the communication protocol is designed, implementation can begin to make the calculations for the desired simulation in the simulation language and send these results through the middleware to the CVRE to make a visual.

For this particular project, all of the information to be extracted from the simulation language already existed in B.I.G. so the only code to implement on the simulation language was the code to grab the job size, type, and method selected by the user, prepare this information into a json file format, and pass the message to the CVRE in a PHP format.

Similar to B.I.G. code existed in O3D that could be used to help create the visualization calculated by the simulation language. However, code to handle the message received from B.I.G. in json format had to be implemented. This required receiving the message, interpreting the message to figure out what excavation was to be made, the workers and/or equipment to be displayed, the type of workers, and the location of each in relevance to the shape of the excavation.

The last code which needed to be implemented was the middleware, PHP. This was simple as the messages coming from and going to the simulation language and CVRE were previously determined and an example of how to pass this data was also implemented in an earlier step.

Figure 10 shows an example of one exercise completed by a student where O3D has been embedded in B.I.G.

Fig. 10 Visual of an apartment job selected by a student with a job size of $19,462 and excavation method 2.

Step 7: *Test the simulation in the CVRE*

In order to test the implemented simulation Professor Hal Johnston was contacted via a conference call where he was given a tutorial walk-through. After the tutorial was complete, he was allowed to experiment on his own and provide feedback of his experience. The feedback provided was very positive in regards to embedding the CVRE into B.I.G. as the only change requested was some rearrangement of the buttons and a changing of words on a particular button. In addition to these requests, Professor Johnston also provided a letter of support where he said "I believe that Lacey's dissertation work is real progress towards richer educational and training materials built upon and extending the capabilities of existing back end simulations such as B.I.G."

V. CONCLUSION

Through the use of a communication protocol methodology developed by Lacey Duckworth the CVRE, O3D, was embedded into an existing simulation language, B.I.G. This implementation allowed for a non-ad-hoc design of the code which would allow other methods of B.I.G. including the basement, foundation, etc. to quickly be implemented. In order to show the simplicity of adding the figures 11 and 12 and Table 3 shows changes that need to be made to the state diagram, transition table, a portion of the communication protocol in order to begin implementing the addition of this new method requiring simple modifications to the existing code.

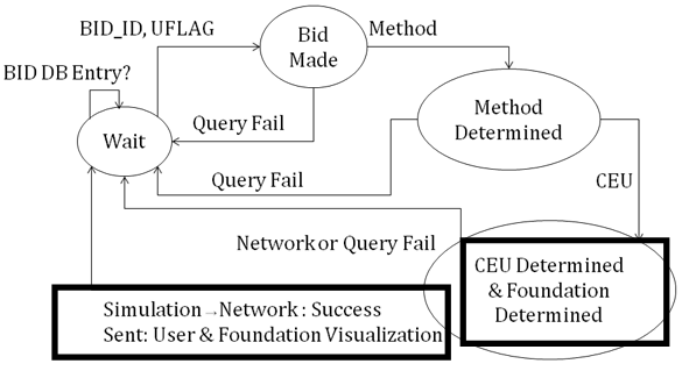


Fig. 11 The state diagram from figure 6 modified to determine the steps in the foundation method

TABLE 3

TRANSITION TABLE DESIGNED USING THE STATE DIAGRAM IN FIGURE 9

	Bid Database Entry	**User, BID_ID, UFLAG**	**Method**	**CEU**	**Simulation -> Network Success Sent: User and foundation Visualization**
Wait	Query for user, Bid_ID, and user flag. Transition to Bid Made	User has made a bid. Query for the user, Bid_ID, and UFLAG.	-	-	-
Bid Made	-	-	Query for CEU	-	-
Method Determined	-	-	-	Determine the foundation to be made	-
CEU and foundation Visualization Determined	-	-	-	-	Send message to simulation adapter and transition to wait stage

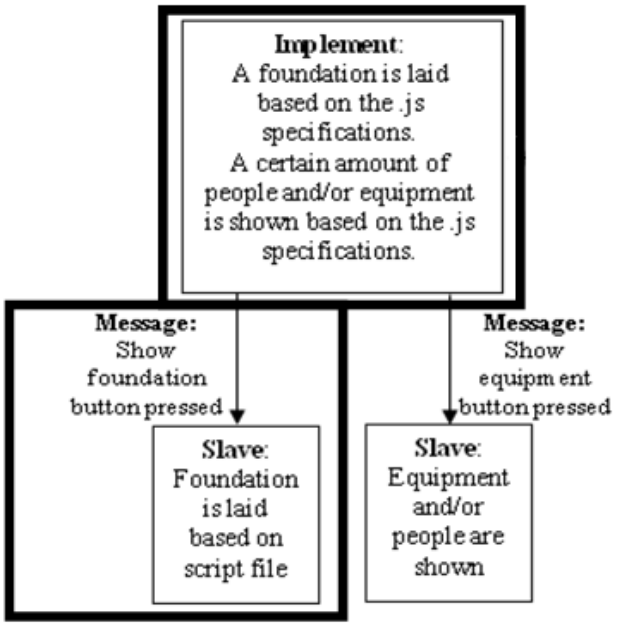


Fig. 12 The communication protocol in figure modified to determine the components to visualize the foundation method in O3D.

This methodology provides a clear design for embedding CVREs into simulation languages as seen in this example. Following this methodology will lead to better designed code and the ability to create more complex by moving the calculations required to maintain such simulations outside of the CVRE.

REFERENCES

[1] H. Johnston, J. Borland, and K. Craig. " Building Industry Game (B.I.G.) A Computer Simulation for Construction Management ," *ASC Proceedings of the 39th Annual Conference*, pp 79-90. [Online]. Available: http://ascpro0.ascweb.org/archives/cd/2003/2003pro/2003/Johnston03.htm

[2] The Unviersity of Southern Mississippi. Open Source Building Environment for Simulation and Training. [Online]. Available: http://www.icee.usm.edu/OSBEST/index.html

[3] Allard, J., Lessage, J. & Raffin, B. (April 2010). Modularity for Large Virtual Reality Applications. Presence : Teleoperators and Virtual Environments. Vol. 19, No. 2, 142-161. doi: 10.1162/pres.19.2.142

[4] Frey, D., Royan, J., Piegay, R., Kermarrec, A.M., Anceaume, E., & Fessant F.L. Solipsis: A decentralized architecture for virtual environments. International Workshop on Massively Multiuser Virtual Environments (MMVE'08), 29 - 33.Retrieved from http://www.pap.vs.uni-due.de/MMVE08/papers/p6.pdf

[5] Hui, L., Jun, Z., Gong, J., Bingli, X., & Hua, Q. (April 2010). A grid-based collaborative virtual geographic environment for the planning of silt dam systems. International Journal of Geographical Information Science, Vol. 24, No. 4, 607-621. doi:10.1080/13658810903012425

First A. Author (M'76–SM'81–F'87) and the other authors may include biographies at the end of regular papers. Biographies are often not included in conference-related papers. This author became a Member (M) of IEEE in 1976, a Senior Member (SM) in 1981, and a Fellow (F) in 1987. The first paragraph may contain a place and/or date of birth (list place, then date). Next, the author's educational background is listed. The degrees should be listed with type of degree in what field, which institution, city, state, and country, and year degree was earned. The author's major field of study should be lower-cased.

The second paragraph uses the pronoun of the person (he or she) and not the author's last name. It lists military and work experience, including summer and fellowship jobs. Job titles are capitalized. The current job must have a location; previous positions may be listed without one. Information concerning previous publications may be included. Try not to list more than three books or published articles. The format for listing publishers of a book within the biography is: title of book (city, state: publisher name, year) similar to a reference. Current and previous research interests end the paragraph.

The third paragraph begins with the author's title and last name (e.g., Dr. Smith, Prof. Jones, Mr. Kajor, Ms. Hunter). List any memberships in professional societies other than the IEEE. Finally, list any awards and work for IEEE committees and publications. If a photograph is provided, the biography will be indented around it. The photograph is placed at the top left of the biography. Personal hobbies will be deleted from the biography.

Development and Evaluation Process of Model Transformation

Shekoufeh Kolahdouz-Rahimi
Dept. of Informatics, Kings College London
Strand, London WC2R 2LS, UK
Email: shekoufeh.kolahdouzrahimi@kcl.ac.uk

Kevin Lano
Dept. of Informatics, Kings College London
Strand, London WC2R 2LS, UK
Email: kevin.lano@kcl.ac.uk

***Abstract*—Model transformations have become a key element of model-driven software development, being used to transform platform-independent models to platform-specific models, to improve model quality, to introduce design patterns and refactorings, and to map models from one language to another. A large number of model transformation notation and tools exist, however, there are no guidelines on how to select appropriate notations for particular transformation tasks.**
In this paper we provide a unified semantic treatment of model transformation, and suggest a possible approach for comparison of model transformation paradigms, on case studies representing the main kinds of model transformation problem (such as refinement, quality improvement and re-expression). The comparison focus on the general property of *comprehensibility*, showing how the Goal/Question/Metric (G/Q/M) work can be used to specialise an approach.

***Keywords*-Model transformations; model transformation specification; UML**

I. INTRODUCTION

Model transformations are mappings of one or more software engineering models (source models) into one or more target models. The models considered may be graphically constructed using graphical languages such as the Unified Modelling Language (UML) [7], or can be textual notations such as programming languages or formal specification languages.

The research area of model transformations remains very active, and certain fundamental issues have as yet only partially been solved:

Specification issues Semantically, model transformations can be considered to be relations between (the sets of models of) languages, ie., in the binary case, they identify for a pair (*M1*, *M2*) of models of two languages, if *M1* is related by the transformation to *M2*.

But for convenience and comprehensibility of specification, transformations are usually defined by sets of transformation rules which relate specific elements of *M1* to specific elements of *M2*. This introduces problems of dependency and consistency between rules: for instance, one rule may read data created by another rule, so must be applied after it. Different orders of application of rules to a source model may result in different target models, so that a model transformation is not uniquely specified by a set of element-to-element rules.

In general, the overall effect of the set of rules may be difficult to deduce from the rules themselves, either for a human reader of the specification, or for specification analysis and verification tools [16].

Development issues At present, construction of model transformations is focussed upon the implementation of a transformation in a particular model transformation language [22], the transformation is described only at a relatively low level of abstraction, without a separate specification. The development process may be ad-hoc, without systematic guidelines on how to structure transformations. The plethora of different transformation languages creates problems of migration and reuse of transformations from one language to another.

Ironically, the model transformation community has recreated the development problems, for this specialised form of software, which model-driven development (MDD) was intended to ameliorate: the inability to reuse and migrate systems due to the lack of platform-independent specifications, and an excessive implementation focus [4].

Implementation issues Declarative specifications of transformations need to be implemented in an efficient manner, and implementations of a transformation need to be shown to be correct with respect to its specification.

Implementations may need to manage traces of the transformations applied, i.e., records of which elements in the target model are related, by which rules, to which elements in the source model.

The solution which we propose to these problems is to adopt a general model-driven software development approach, UML-RSDS (Reactive System Development Support), for model transformation development. UML-RSDS provides the necessary specification notations for model transformation definition, at different levels of abstraction, and provides support for verification and the synthesis of executable code. Transformations can be specified as global relations, and then decomposed into phases composed of individual rules, the correctness of the decomposed specification with respect to the global relation can then be

demonstrated. UML-RSDS uses standardised UML notation and concepts, so permitting reuse and interchange of models. Existing semantics for UML [5] can be adopted to give a semantics for model transformations defined in UML-RSDS.

We consider refinement, re-expression and quality improvement transformations, based on the relative levels of abstraction of the source and target languages, and the semantic relation between source and target models.

Refinement transformations map models in one language to semantically stronger models in the same or a different language. Examples include PIM to PSM mappings in the MDA [8], or the introduction of design patterns such as Singleton.

Re-expression model transformations map models in one language to semantically equivalent models in a different language. Examples could be model migration from one version of a metamodel to another [15].

Quality improvement transformations map models into semantically equivalent models in the same language. Examples are UML class diagram refactorings [16] or the introduction of design patterns such as Template Method.

Several feature of model transformation are discussed in [23], however the *Comprehensibility* feature of model transformation is not considered. Comprehensibility, refers to the ability in which specification and implementation of model transformation can be understood and analysed. A comprehensive specification is crucial for an efficient maintenance process [24].

Measurement is the process of detecting and recording the observations that are collected as part of the research attempt. We will analyse the comprehensibility feature of different model transformation languages with the popular Goal/Question/Metric (G/Q/M) paradigm [25]. G/Q/M paradigm, direct the data toward a measurable goal and every goal should be answered by one or more key questions. The metrics address the goal or part of the goal by answering specific questions.

II. Specification Techniques for Model Transformations

A large number of formalisms have been proposed for the definition of model transformations: the pure relational approach of [1], graphical description languages such as graph grammars [26] or the visual notation of QVT-Relations [QVT], hybrid approaches such as Epsilon [27] and implementation-oriented languages such as Kermeta [21].

In each approach, model transformations are specified and implemented as rules specific to particular kinds of elements in the source model(s) and individual elements in the target model(s). The model-to-model relation is then derived from some composition of these individual rules.

This raises the question of how it can be shown that the composition of the rules achieves the intended global relation, and what semantics should be used to carry out the composition. In the case of languages such as ATL [11] and QVT- Relations, the order of invocations of rules may be only implicitly defined, and may be indeterminate.

III. Transformation Specification in UML-RSDS

UML-RSDS is a UML-based specification language, consisting of UML class diagrams, state machines, activities, sequence diagrams, and a subset of OCL. It is used as the specification language of an automated Model-Driven Development approach, Constraint-Driven Development (CDD) [6], by which executable systems can be synthesised from high-level specifications.

The most abstract form of specification in UML-RSDS consists of class diagrams together with constraints, the constraints serve to define both the static state and dynamic behavior of the system, and executable code is synthesized from such constraints using the principle that any operation *op* that changes the state of the model must have an executable implementation which ensures that all the model constraints remain true in the modified state.

It is also possible to use a lower level of abstraction, and to explicitly specify operations using pre and post-conditions in OCL, and define activities or state machines to specify the order of execution of operations within a class, or of individual steps of specific operations.

IV. Development Process for Model Transformations

Our general recommended development process for model transformations is as follows:

Requirements The requirements of the transformation are defined, the source and target metamodels are specified, including which constraints need to be established or preserved by the transformation, and what assumptions can be made about the input models. A use case diagram can be used to describe the top-level capabilities of the system, and non-functional requirements can be identified.

Abstract Specification Constraints can be used to define the overall relation *Rel* between source and target models for each use case (transformation). We will usually express the precondition of a transformation (considered as a single operation) as a predicate *Asm*, and the postcondition as a predicate Cons, both *Asm* and Cons may be expressed in the union of the languages of the source and target models. *Asm* defines the domain of *Rel* , and Cons defines which pairs of models are in *Rel* . Informal diagrams in the concrete syntax of the source and target languages can be used to explain the key aspects of this relation, as in the CGT model transformation approach [4].

Explicit Specification and Design The transformation can be broken down into phases, each with its own source and target language and specification. Phases should be independent of each other, except that the assumptions of

a non-initial phase should be ensured by the preceeding phase(s).

For each phase, define transformation rules (as operations specified by pre/postconditions), and an activity to specify the order of execution of the rules. Recursion between rules should be avoided if possible. Again, informal diagrams can supplement the formal definition of the rules. For each phase, verification that the activity combination of the rules satisfies the overall specification of the phase can be carried out. It can also be checked that the rule operations are deterministic and well-defined, and that the activities are confluent and terminating under the precondition of the phase. Finally, it should be checked that the composition of the phases achieves the specified relation Cons and required property preservation/establishment conditions of the overall transformation:

V. Case Studies

We will compare transformation specification approaches on three case studies, which represent each of the three kinds of transformation we are considering:

1) Tree to graph: a re-expression transformation.
2) UML to relational database schemas: a refinement transformation.
3) Class diagram refactoring: a quality improvement transformation.

Within these case studies we use some common-used model transformation tools and apply a GQM plan to them for evaluation of comprehensibility feature.

A. case study 1: Tree to Graph Transformation

Figure 1 shows the source and target metamodels of the transformation from trees to graphs. The tree metamodel has the language constraint that there are no non-trivial cycles in the $parent$ relationship:

$$t : Tree \ and \ t \neq t.parent \ \ implies \ \ t \notin t.parent^+$$

where r^+ is the non-reflexive transitive closure of r. Trees may be their own parent if they are the root node of a tree.

Semantic correctness requires that this condition remains true, in interpreted form, in the graph resulting from applying the transformation to a tree.

The graph metamodel has the constraint that edges must always connect different nodes:

$$e : Edge \ \ implies \ \ e.source \neq e.target$$

and that edges are uniquely defined by their source and target, together:

$$e1 : Edge \ and \ e2 : Edge \ and$$
$$e1.source = e2.source \ and \ e1.target =$$
$$e2.target \ implies \ e1 = e2$$

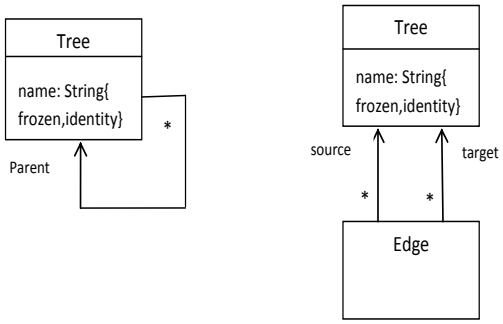


Figure 1. Tree to graph transformation metamodels

The $identity$ constraint means that tree nodes must have unique names, and likewise for graph nodes.

The transformation relates tree objects in the source model to node objects in the target model with the same name, and defines that there is an edge object in the target model for each non-trivial relationship from a tree node to its parent.

The case study has implemented in five common-used transformation languages.

In this example, the mapping from trees to graph nodes could be expressed by two operations in UML-RSDS:

$mapTreeToNode(t : Tree)$ `post:`
$\exists \ n : Node \cdot n.name = t.name$

$mapTreeToEdge(t : Tree)$ `pre:` $t \neq t.parent \ \ and$
$t.name \in Node.name \ \ and$
$t.parent.name \in Node.name$
`post:`
$\exists \ e : Edge \cdot$
$e.source = Node[t.name] \ \ and$
$e.target = Node[t.parent.name]$

The notation $Node[x]$ refers to the node object with primary key (in this case name) value equal to x, it is implemented in the UML-RSDS tools by maintaining a map from the key values to nodes. In OCL it would be expressed as

$$Node.allInstances() \rightarrow select(name = x) \rightarrow any()$$

Likewise, $Node.name$ abbreviates

$$Node.allInstances() \rightarrow collect(name)$$

No changes were needed to the source or target metamodels. UML-RSDS also supports the definition of metamodel constraints, such as the no-cycles property of the tree metamodel.

A *ruleset* in UML-RSDS is a set of rules (operations), it is defined as a UML class with a behaviour defined by an activity. This controls the allowed order of application of the rules. In this example this order is to iterate all the tree-to-node mappings first, then the tree-to-edge mappings:

$\texttt{for } t : Tree \texttt{ do } mapTreeToNode(t) \texttt{ ;}$
$\texttt{for } t : Tree \texttt{ do } mapTreeToEdge(t)$

Likewise in the graph to tree direction, a similar ruleset can be defined, consisting of rules $mapNodeToTree(n : Node)$ and $mapEdgeToTree(e : Edge)$ which define a tree and its $parent$ association from a graph.

B. Case stusy 2:The UML to relational database model transformation

This case study concerns the mapping of a data model expressed in UML class diagram notation to the more restricted data modelling language of relational database schemas. Modelling aspects such as inheritance, association classes, many-many associations and qualified associations need to be removed from the source model and their semantics expressed instead using the language facilities (tables, primary keys and foreign keys) of relational databases.

We consider the published specifications of this problem for VIATRA [12], ATL [11], QVT-R [9] and Kermeta [10], and our own version defined in UML-RSDS. We compare these by applying metrics of size and complexity to the specifications.

C. Case study 3:Class diagram rationalisation

This case study is a typical example of an update-in-place quality improvement transformation. Its aim is to remove from a class diagram all cases where there are two or more sibling or root classes which all own a common-named and typed attribute.

It is used as one of a general collection of transformations (such as the removal of redundant inheritance, or multiple inheritance) which aim to improve the quality of a specification or design level class diagram.

Figure 2 shows the metamodel for the source and target language of this transformation.

It can be assumed that no two classes have the same name, no two types have the same name, The owned attributes of each class have distinct names, and do not have common names with the attributes of any superclass, There is no multiple inheritance

These properties *Asm* must also be preserved by the transformation.

The informal transformation steps are the following:

- If a class *c* has two or more immediate subclasses *g* = *c.specialization.specific*, all of which have an owned attribute with name *n* and of type *t* , add an attribute of this name and type to *c* and remove the copies from each element of *g*.

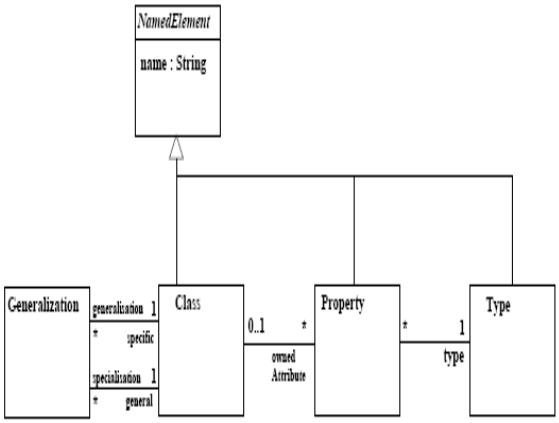


Figure 2. Basic class diagram metamodels

The formal version of rule one in UML-RSDS is as follows:

$rule1(\ c : Class,\ n : String,\ t : Type\ )$
$\texttt{pre:}$
$c.specialization->size() > 1 \ and$
$c.specialization.specific->$
$forAll(ownedAttribute->exists$
$(p \mid p.name = n \ and \ p.type = t))$

$\texttt{post:}$
$(p : Property . p.name = n \ and$
$p.type = t \ and \ p : c.ownedAttribute)$
and
$c.specialization.specific->forAll($
$c1.ownedAttribute = c1.ownedAttribute$
$@pre->reject(p \mid p.name = n))$

- If a class *c* has two or more immediate subclasses *g* = *c.specialization.specific*, and there is a subset *g1* of *g*, of size at least 2, all the elements of which have an owned attribute with name *n* and of type *t* , but there are elements of *g* without such an attribute, introduce a new class *c1* as a subclass of *c* and as a direct superclass of all those classes in *g* with the attribute. Add an attribute of this name and type to *c1* and remove from each of its direct subclasses.

The formal version of the second rule in UML-RSDS is as follows:

$rule2(c : Class,\ n : String,\ t : Type)$
$\texttt{pre:}$
$c.specialization->size() > 1 \ and$
$c.specialization.specific->select($
$ownedAttribute->exists(p \mid p.name = n$
$and \ p.type = t))->size() > 1 \ and$
$not(c.specialization.specific->forAll($
$ownedAttribute->exists(p \mid p.name = n$
$and \ p.type = t)))$
$\texttt{post:}$
$\exists\ c1 : Class\ .\ c1.name = c.name$

$$
\begin{array}{l}
\quad + \text{"\_"} + n \quad and \\
(\exists\ g : Generalization\ .\ g1.general = c \\
\quad and\ g1.specific = c1\ and \\
\exists\ p : Property\ .\ p.name = n \quad and \\
\quad p.type = t \quad and \\
\quad p : c1.ownedAttribute) \quad and \\
\quad c.specialization.specific-> \\
\quad select(ownedAttribute-> \\
\quad exists(name = n\ and\ type = t)-> \\
\quad forAll(c2 \mid c2.ownedAttribute = \\
\quad c2.ownedAttribute@pre-> \\
\quad reject(name = n)\ and \\
\exists\ g1 : Generalization\ .\ g1.general = c1 \\
\quad and\ g1.specific = c2)
\end{array}
$$

- If there are two or more root classes all of which have an owned attribute with name *n* and of type *t* , create a superclass *c* of all such classes and add an attribute of this name and type to *c* and remove from each of its direct subclasses.
  The formal version of the thired rule in UML-RSDS is as follows:

$$
\begin{array}{l}
rule3(n : String,\ t : Type) \\
\ \texttt{pre:} \\
\quad Class.allInstances()-> select \\
\quad (generalization-> size() = 0)-> \\
size() > 0 \\
\quad and \\
\quad Class.allInstances()-> select \\
\quad (generalization-> size() = 0)-> \\
\quad select(ownedAttribute-> exists \\
\quad (name = n \quad and \quad type = t)) \\
\quad -> size() > 1 \\
\texttt{post:} \\
\quad \exists\ c1 : Class\ .\ c1.name = \\
\quad \text{"Root\_"} + n \quad and \\
\quad (\exists\ p : Property\ .\ p.name = n \\
\quad and \quad p.type = t \quad and \\
\quad p : c1.ownedAttribute) \quad and \\
\quad Class.allInstances()-> select \\
\quad (generalization-> size() = 0)-> \\
\quad select(ownedAttribute-> \\
\quad exists(name = n\ and\ type = t)-> \\
\quad forAll(c2 \mid c2.ownedAttribute = \\
\quad c2.ownedAttribute@pre-> \\
\quad reject(name = n)\ and \\
\ \exists\ g1 : Generalization\ . \\
\quad g1.general = c1 \\
\quad and\ g1.specific = c2)
\end{array}
$$

It is required to minimise the number of new classes introduced, ie, to priorities rule 1 over rules 2 or 3.

The key tasks for this problem are:

- Provide a global declarative specification of the transformation which describes its effect as a relation *Cons* between the source and target models.
- Show that this relation establishes the required properties of the target model (the properties to be preserved, *Asm*).
- Define an implementation of the transformation, using the informal rules or in some other manner.
- Analyse the confluence and termination properties of this implementation. There should be an natural number valued quality measure *Q* of a model which is decreased by each transformation step, and reaches zero when no step can be applied.
- Show that the implementation satisfies *Cons*.

Additional tasks are to consider ways to implement change-propagation, and to extend the transformation to models with associations to also remove duplicated association end properties of sibling classes.

VI. Evaluation of Comprehensibility in Model Transformation

The purpose of this section is to apply G/Q/M paradigm to the case studies, and tailor into a powerful measurement framework for model transformation. The framework enables us to consider comparative evaluation of transformations approaches from the most abstract perspective.

The first step in this framework is to establish the goals of the data collection. *Comprehensibility* is considered as the main goal for this research.

To assess the goal, a list of questions of interest have been developed. The first question that was asked to evaluate the goal is : "What is the overall size of transformation". The more lines of code in the specification, the more effort is needed to understand and analyse the transformation. The development of second question was based upon the complexity of transformation. Lack of simplicity in the specification affects the understandability of transformation. Another possible question that achieves the goal, is associated with modularity. The comprehensibility of transformation is enhanced by defining strategies to encapsulate sets of rules inside each module.

Once the questions were developed, the metrics are tailored to answer the question and address a goal or part of a goal. The snapshot of G/Q/M plan for comprehensibility in model transformation is shown in figure 3.

Different programming styles may require different counting techniques which can lead to different measurements. However, an obvious size metric is the number of lines of code in transformation specification or implementation.

Complexity of a transformation is influenced by calling relationship between subroutines in a program. In order to understand a rule, it is also needed to know the rules it calls. On the other hand, recursion calls have substantial effect on complexity which makes the chain more complex.

It is therefore likely that the total number of calls, recursion as well as non-recursion calls in transformation be counted. Depth of call chains also influence the complexity. The deeper the call chain, the harder it is to understand the transformation.

The possible answer to modularity question is to count the sum of interconnectivity between distinct modules (Cou-

pling) as well as the sum of interconnectivity inside each individual module (Cohesion). Low coupling results in less dependency between modules and therefore enhances the level of understanding. High cohesion implies that all the modules internal statements serve to perform a single task and therefore high level of understandability.
The next step involves collection and validation of data. Therefore the metrics were applied to the case studies and evaluation on collected data is processed.

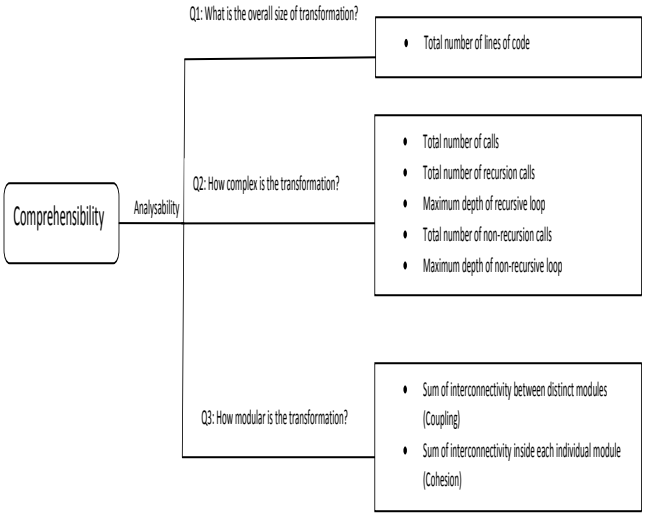


Figure 3. G/Q/M Snapshot

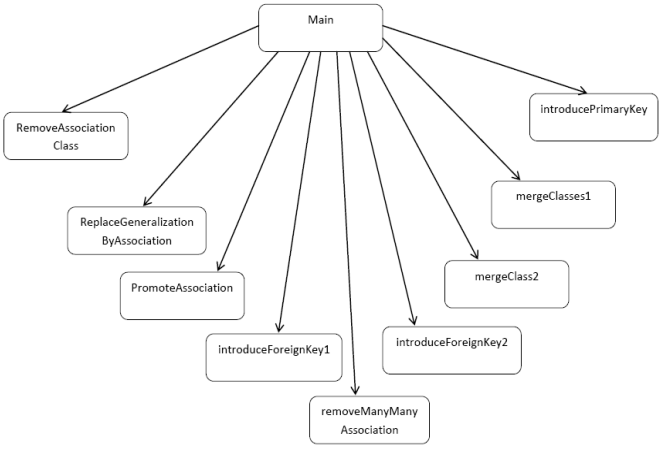


Figure 4. Call graph of UMLRSDS in uml to relational database schema transformation

Table I
COMPLEXITY OF UML TO RELATIONAL DATABASE TRANSFORMATIONS

Approach	Total number of calls	Total recursive calls	Depth2	Total non-recursive calls	Depth1
Kermeta	22	9	2	13	4
VIATRA	16	0	0	16	3
QVT	10	4	2	6	4
UML-RSDS	11	0	0	11	1
ATL	6	0	0	6	2

1) Complexity: of a specification can be measured by analysing its call graph. A call graph is a directed graph that represents calling relationships between subroutines in a program. In the call graph each node represents functions or rules and edges represent calls between nodes. The size of the call graph affects the complexity of the program. The greater the number of arcs in the call graph, the higher is the dependency between different parts of the program, and so the greater is the complexity.

Figure 4 shows the call graphs of the UML-RSDS specifications on second case study. Each node represents a rule and edges represent the calls from one rule to another, both implicit and explicit calls.

Table I compares the complexities of the approaches, based on the total number of calls and depth of calls on second case study. Depth1 is the maximum depth of call chains not involving recursive loops, and Depth2 the maximum length of a recursive loop.

This shows considerable differences in the styles of specification adopted, with recursion being used substantially in some solutions (Kermeta and QVT), and not used at all in the other solutions. The problem does naturally lead to a recursive solution, because of the recursive structure of class diagrams (mapping a class to a relational table involves mapping its super or sub-classes, and its owned attributes, which may be of a class type). However a phased solution is also possible, where basic elements (such as attributes of non-class type, and classes without subclasses) are mapped before elements composed from these elements. Such a phased solution could be defined in Kermeta. The QVT solution contains a potentially unbounded recursion (mapping attributes of a class type in the case of mutual dependencies between two classes), and only specifies local ordering restrictions between rules (for example, the classes at either end of an association must be mapped before the association itself). This provides greater flexibility in execution order than a fixed scheduling of rules, and hence improves the potential for optimizing execution efficiency. Furthermore, this makes the proof of confluence and correctness of the transformation more difficult.

The strategy for evaluation of complexity in quality improvement case study involves measurement of expressions in the specification.The number of operators and number of distinct features/variables in expression is counted. For

instance the specification of rule 1 in UML-RSDS contains 14 operators in precondition and 20 in the postcondition. In addition, number of distinct variable is 9 in precondition and is 10 in postcondition.

VII. Conclusion

Development approaches for model transformations have been formulated in [22], however most model transformation development remains focused upon the implementation level. Ideally, model-driven development should be applied to model transformations, with verification of the correctness and consistency of the transformations being carried out as an integral part of such development. We have defined a development process and specification technique for model transformations, using UML-RSDS .

The process measurement framework in this paper helps by providing specific goals, questions and metrics. Metrics evaluate complexity of different model transformation approaches on two case studies. G/Q/M is a useful approach for identifying relevant metrics for model transformation. The goals provides suitable framework for the analysis and interpretation of collected data. We found out that QVT and Kermeta have large complexity in terms of the number of recursive calls which hindered the verification.

References

[1] Akehurst, D. H. and Kent, Stuart, *A Relational Approach to Defining Transformations in a Metamodel*, in UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language, pp. 243–258, Springer-Verlag,2002.

[2] J. Cuadrado, J. Molina, *Modularisation of model transformations through a phasing mechanism*, in Software Systems Modelling,Vol. 8, pp.325345, 2009.

[3] I. Kurtev and K. G. van den Berg and F. Jouault, *Rule-based modularization in model transformation languages illustrated with ATL*, in Science of computer programming, Vol. 68, pp. 111–127, 2007.

[4] R. Gronmo, B. Moller-Pedersen, G. K. Olsen, *Comparison of Three Model Transformation Languages*, in proceedings of ECMDA-FA,LLNCS 5562,pp. 217,2009.

[5] K. Lano (ed.), *UML 2 Semantics and Applications*, in Wiley,2009.

[6] K,Lano, *A Compositional Semantics of UML-RSDS*, in SoSyM, vol. 8, no. 1, pp. 85116, February2009.

[7] OMG, *UML superstructure, version 2.1.1. OMG document formal/2007-02-03*, 2007.

[8] OMG, *Model-Driven Architecture*, http://www.omg.org/mda/,2004.

[9] OMG, *Query/View/Transformation Specification*, 2009.

[10] Kermeta, http://www.kermeta.org, 2010.

[11] F. Jouault, I. Kurtev, *Transforming Models with ATL*, in MoDELS 2005, LNCS Vol. 3844, pp. 128–138, Springer-Verlag, 2006.

[12] OptXware, *The Viatra-I Model Transformation Framework Users Guide*, 2010.

[13] S. Kolahdouz-Rahimi, *Model Transformation Specification in UML-RSDS*, ICST PhD Symposium, 2010.

[14] Krzysztof Czarnecki and Simon Helsen *Classification of Model Transformation Approaches*, OOPSLA03 Workshop on Generative Techniques in the Context of Model-Driven Architecture, OOPSLA 2003

[15] K.Lano, S.Kolahdouz-Rahimi, *Model Migration Transformation Specification in UML-RSDS*, TTC2010, Malaga, Spain, 2010.

[16] S. Markovic, T. Baar, *Refactoring OCL Annotated Class Diagrams*, MoDELS 2005, Springer-Verlag LNCS vol. 3713, Springer-Verlag, 2005.

[17] P. Stevens, *Bidirectional model transformations in QVT*, SoSyM vol. 9, no. 1, 2010.

[18] OMG, *Meta Object Facility (MOF) Core Specification*, OMG document formal/06-01-01, 2006.

[19] K. Lano, *Constraint-Driven Development*, Information and Software Technology, 50, 2008, pp. 406–423.

[20] F. Jouault, F. Allilaire, J. Bezivin, I. Kurtev, *ATL: A model transformation tool*, Science of Computer Programming, 72, pp. 31–39, 2008.

[21] Z. Drey, C. Faucher, et. al., *Kermeta Language Reference Manual*, http://www.kermeta.org/docs/KerMeta-Manual.pdf, April 2009.

[22] E. Guerra, J. de Lara, D. Kolovos, R. Paige, O. Marchi dos Santos, *transML: A family of languages to model model transformations*, MODELS 2010, LNCS vol. 6394, Springer-Verlag, 2010.

[23] Kevin Lano, Shekoufeh Kolahdouz Rahimi, *Specification and Verification of Model Transformations Using UML-RSDS* ,in IFM, PP.199-214, 2010.

[24] Rozilawati Razali and Paul W Garratt, *Measuring the Comprehensibility of a UML-B model and a B model*, in International Conference on Computer and Information Science and Engineering CISE 2006, pp.338–343, 2006.

[25] Heiko Koziolek, *Goal, Question, Metric*, Dependability Metrics, pp.39-42,2005.

[26] Ehrig, H. and Engels, G. and Kreowski, H.-J. and Rozenberg,G., *Handbook of graph grammars and computing by graph transformation: vol. 2: applications, languages, and tools*,World Scientific Publishing Co., Inc.,1999.

[27] Dimitrios S. Kolovos and Richard F. Paige and Fiona Polack, *The Epsilon Transformation Language*,in ICMT, pp.46-60, 2008.

Modeling and Analysis of Agent Oriented System: Petri Net Based Approach

Rajib Kr. Chatterjee[1], Anirban Sarkar[2], Swapan Bhattacharya[3]
[1]Department of Computer Centre, National Institute of Technology, Durgapur, India
[2]Department of Computer Applications, National Institute of Technology, Durgapur, India
[3]Department of Computer Science & Engineering, Jadavpur University, Kolkata, India
{chatterjee.rajib@gmail.com , sarkar.anirban@gmail.com , bswapan2000@yahoo.co.in}

Abstract - *Large and complex system are now a days conceptualized using Agent Oriented Paradigm. Agent oriented systems are dynamic in nature. In this paper, we have proposed a conceptual framework for agent to conceptualize the artifacts of such system. The paper also has proposed a Petri Net based model and analysis methodology based on that conceptual framework to analyze the crucial behavioral feature of such system which is also dynamic in nature.*

Keywords: Agent, Agent Oriented System, Petri Net, Dynamic Modeling, Behavior Analysis.

1 Introduction

Over the last decades, software engineers have derived a progressively easy and better perception of the characteristics of large and complex software. Among others, such software is characterized by dynamically interacting components to provide wide range of services. In this context, Agent Oriented System (AgOS) recently emerged as powerful technology to handle the dynamism of component level interactions for large and complex information system. AgOS based computing promotes, designing and developing applications in terms of autonomous software entities (agents), situated in an environment, and that can flexibly achieve their goals by interacting with one another dynamically in terms of high-level protocols or languages [3].

In large information system, agent refers to a software component that situates within some environments, operates autonomously and cooperates with similar entities to achieve a set of preset goals. An agent also may associate with its mental state that can be composed of components like belief, knowledge, capabilities, choices and commitments [6]. The critical features of agent are as follows,

- *Autonomous:* Agent is composed of some predetermined states and is able to take any decisions based on those states without any direct intervention of actors of the environment.
- *Goal oriented:* Agent always acts or works to achieve or reach the preset target or goal. If an AgOS composed of multiple agents then they together can achieve the goal through the cooperative activities.
- *Capabilities:* Each agent is capable to perform certain activities towards achieving the goal. These activities are often characterized by set of well defined services which may be provided by the agent. The agent capabilities can be defined using these set of services.
- *Situatedness:* An agent performs its activities while situated in a particular environment and it is able to sense and affect such environment.
- *Proactive/Reactive:* Agent not only acts in response to the events of the environment where it is situated but they may also become active autonomously. Besides this proactive nature, agent may act dynamically to understand the environment, apprehend any changes in this environment and respond timely to the changes that may occur.
- *Knowledge Driven:* An agent can have the ability to acquire new knowledge about the environment in which it is deployed and can update dynamically.
- *Condition/Constraint:* The environmental constraints may affect the activities of agent. Moreover such constraint may be imposed by actors of the environment and which can be adopted dynamically by the agent.

Several of these features have been summarized in recent literatures [1, 3, 6]. Along with these agent level characteristics, the crucial features for AgOS can be summarized as follows,

- *Agent Social:* An AgOS may be comprised of multiple agents which are supposed to operate together in an open operational environment. Hence they can interact with each other, share their resources and knowledge, and also can collaborate with each others to achieve the preset goals.
- *Resource Driven:* Agent acts on environmental resources. Any agent of AgOS can hold, use and release resources of the environment where it is situated. The activities can change the state of the resources to fulfill predetermined goal or objective.
- *Event Driven:* Agents of AgOS response on events occurred in the environment. Events may occur due to some state

changes or achieving certain condition or achieving certain goal or certain interaction of actors. Even more events may occur due to certain changes in environment. An AgOS achieve any goals using a series of events occurrences and the ongoing events may determine the system behavior.

- *Dynamic:* Due to event driven nature of AgOS and with the feature like autonomous and reactiveness of agent, such systems are truly dynamic. Moreover, the knowledge of any agent can be dynamic in nature. Further, the series of events and its corresponding responses may occur dynamically from such system. Designer simply set the initial state, knowledge and goals, on next, AgOS manage the things dynamically to achieve the goal.
- *Heterogeneous:* Several agents of AgOS may be heterogeneous in nature in terms of their features. They may initially belong to different environment. Any agent may be reused to cooperate with other set of agents to achieve some goals in new environment. These facts require migrating of some specific agent from one environment to another in pre determined fashion. This also characterizes the *mobility* of agent.

While modeling of AgOS, designer must also ensure that, (i) system will achieve the goal with finite number of events or interactions, (ii) system will be able to handle the situation where goal will not be achieved after certain set of events, (iii) system will operate in deadlock free way, as the system will be handling the resources from the environment, (iv) system and environment should transform in acceptable states with the occurrences of events and (v) the knowledge and the state of the resources are dynamically manageable. These all are very crucial features of the dynamic behavior of AgOS. In view of these features, Petri Net [2] is obvious tool choice for modeling the dynamic behavior of AgOS. *Conceptual modeling* of AgOS in this respect, defines the components and their inter relationship to conceptualize the environment, agent, related events and interactions. This specifies the static part of the AgOS. Petri Net based tools will be useful to complement the dynamic part of such system and analyze the states and behavior of agents in the environment.

Several researches in last decade have been done to devise conceptual model for AgOS [4, 5, 6, 7, 8, 9, 11]. Among those proposed approaches, [4, 6, 9, 11] have extended the Unified Modeling (UML) notations to conceptualize the AgOS using object oriented paradigm. In [10], a detail study has been done on these proposed approaches and raised the demand of new paradigm beyond the object oriented paradigm to conceptualize the AgOS. It also states that agent architecture is far more complex than the object architecture, especially because of the dynamic aspects of AgOS. But majority of those proposed approaches have not been dealt with the dynamic behavior of AgOS. In [5], a formal frame work for AgOS has been devised using ObjectZ notation and the semantic of behavior has been represented using state chart diagram. But it lacks the methodology for dynamic behavior analysis of such model, which is crucial for the successful deployment of AgOS.

In [12], a high level Petri Net has been defined to formalize the AgOS. It is efficient to model the external behavior (interaction with the environment) of the AgOS comprises with homogeneous set of multiple agents. But it lacks to exhibit the dynamism of internal behavior (within the agents) of the system which comprises of heterogeneous set of agents. Moreover, several crucial properties related to dynamic behavior have not been analyzed using the proposed high level Petri Net.

The focus of this paper is two folds. *Firstly*, it proposes a conceptual framework for Agent and AgOS to conceptualize its artifacts. *Secondly*, based on the proposed conceptual framework, the dynamic behavior analysis of AgOS has been done using classical Petri Net. For the purpose, we have proposed a generic Petri Net representation for the conceptual framework of agent. Moreover, Petri Net based analysis has been used to ensure the correctness of the crucial characteristics of dynamic behavior of the agent.

2 Proposed Conceptual Framework for Agent

In this section a conceptual framework for AgOS has been proposed. A conceptual model of AgOS deals with high level representation of the candidate environment in order to capture the user ideas using rich set of semantic constructs and interrelationship thereof. Such conceptual model will separate the intention of designer from the implementation and also will provide a better insight about the effective design of AgOS. The framework has been drawn from the system features discussed in the last section.

An environment *Env* where the agents will work can be realized using four tuples. It can be defined as *Env = [Res, Actor, Agent, Relation]*, where, in the given environment *Res* is the set of resources, *Actors* are the users, *Agent* is the set of autonomous entities with pre specified goal and *Relation* is set of semantic association among them.

In the context of *Env*, an agent will apprehend the occurrences of events automatically and response towards the environment with a set of activities or services, those are within its capability. An agent will also be able to verify the environmental conditions / constraints associated with the services or occurrence of any events. Moreover, any agent acts on the environmental resources *Res* and is able to create / maintain the knowledge base for the states of resources. The states of agent can be realized using a set of attribute associated with that agent.

Formally an agent *Ag* in the environment *Env* can be defined as, *Ag, = [E, C, R, PR, K, S]* where,

- *E* is the set of events $\{e_1, e_2, e_3, \ldots, e_p\}$ on which the agent will response. The events may occur from the *Actor* or changes in states of *Res* or on achieving some condition.

- *C* is a set of environmental conditions or constraints $\{c_1, c_2, \ldots, c_q\}$ to be checked in order to response on some event.
- *R* is a set of environmental resources those are available and necessary for fulfillment of the goal of the agents. Also $R \subset Res$.
- *PR* is the set of agent properties which will hold the state of the agent and also will maintain the state of the resources *R* on which the agent is acting.
- *K* is the set of information that forms the main knowledge base. Initially it comprises of the states of available resources that the agent will use to response on some event. The *K* can be updated dynamically.
- *S* is the set of services that the agent can provide. The set *S* is used to conceptualize the capability of the agent. The agent may provide the service $s_i \in S$ to the environment on the occurrences of some set of events $E' \in E$ to achieve the pre specified goal.

A *Multi Agent System* in this context can be defined as *MAS = [A, I]*, where *A* is the set of agents and *I* is the set of interactions among those agents. The set *I* determines cooperation and collaboration among the agents and also it can operate either in two modes namely, asynchronous or synchronous.

From the above definitions we can represent the conceptual framework of an agent graphically as shown in Figure 1.

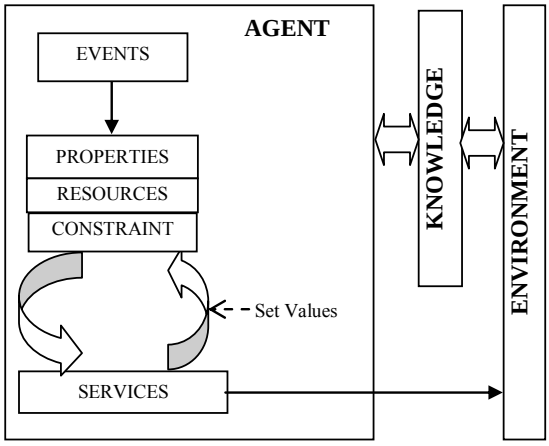


Figure 1: Conceptual Framework of AGENT

2.1 Illustration of Conceptual Framework of Agent with Example

In this sub section we have illustrated the conceptual framework of agent using a real world example. Let consider an environment comprised of set *Computers* with different types of *Operating System* (OS) connected with network. The environment also contains an autonomous *Agent* to act on the *Computers*. After initiation, the goal of the agent is to search the *Computers* with *MS Windows OS* and to shutdown it automatically. The agent will also maintain a time delay of 15 seconds while performing the shutdown activities on the next target *Computer*. Users are allowed to interrupt the agent responses at any point of time. The number of *Computers*, their *OS, Network Addresses (IP addresses)* and the *Port Numbers* (through which agent can communicate with Computer) are well known in the environment.

For the given environment, an agent can be designed using the proposed conceptual framework are given below. For the purpose one need to define all the components of the agent definitions of that said framework to achieve the given goal.

i) The set of events E will be, e_1 = Initiate, e_2 = User Interrupt, e_3 = User Resume, e_4 = User Cancel Job, e_5 = Search, e_6 = Target Computer Found, e_7 = Service Initiate, e_8 = Service Resume, e_9 = Service Completed, e_{10} = Service Interrupt, e_{11} = Service Revoked, e_{12} = Timer start, e_{13} = Terminate, e_{14} = No Action.

Those events may occur after satisfying some environmental constraints C. The agent will response due to some events based on some specific constraint.

ii) The set of constraints C are, c_1 = Time Delay of 15 second, c_2 = Operating System is MS WINDOWS, c_3 = Next IP address to be processed.

After the verification of the constraints, the agent may acts on a set of resources from the environment. For the purpose, it performs some activities on those resources to achieve the goal.

iii) The set of resources R are, r_1 = The network services, r_2 = Computers, r_3 = The OS port where the agent will interact, r_4 = The timer to keep track of the time delay of 15 seconds

Now the agent will use several properties to hold the state of the resources and the states of the agent itself. An agent may changes its state based some events and the state of resources may change based on the activities performed by that agent.

iv) The set of properties PR are, pr_1 = Computers identity with MS Windows OS, pr_2 = OS type of the current Computer, pr_3 = Time Elapsed. Its initial value will be 0, pr_4 = Status type of the agent and it can be of the following types, a) "INIT", b) "INTERRUPT", c) "COMPLETED", d) "CANCELLED" and e) "RESUME".

Agent starts working with the minimal set of knowledge of the environment to render the services. The knowledge base accumulates the initial facts of the resource states which are static in nature. The knowledge base can be updated dynamically once the agent starts working.

v) The set of knowledge K are, k_1 = IP Addresses list of the Computers, k_2 = OS of the Computers, k_3 = Selected Port Numbers of the Computers.

To achieve the pre specified goal, agent acts on environmental resources with certain activities. These activities are realized using a set of services.

vi) The set of services S are, s_1 = Seek OS type from the computer, s_2 = Action Shutdown, s_3 = Action Paused (Act on agent itself), s_4 = Action Revoked (Act on agent itself), s_5 = Seek Port Number, s_6 = Set Port Number, s_7 = Action Terminate (Act on agent itself), s_8 = Action resume (Act on agent itself).

As a result of the triggered events, some set of services will be performed by the agent based on the certain values of the properties and constraints. It will use the resources which are all accessible from the knowledge base. Thus with all these components a generic and autonomous agent will be able to perform the pre specified goal to shutdown all the *Computers* with *MS Windows OS* in the given environment.

3 Petri Net based Modeling of Conceptual Agent

AgOS behavior is dynamic in nature. Also several crucial features are required to ensure while designing such system. Since, agent response on series of events and provide services to the environment in autonomous way. For the purpose it holds the resources and makes changes in its states. Petri Net (PN) is a suitable tool to model the behavior of such system. Moreover, several features of AgOS like, occurrence of finite number of events, deadlock free operations, achievement of goals through firing of events etc. can be analyzed through the analysis of PN properties like, safeness, boundedness, liveness, reachability etc. Further, the PN based analysis will give detail insight about the internal behavior of the agent.

In this context, a PN is a particular kind of bipartite directed graph, populated by three kinds of objects namely, places, transitions and directed arcs connecting places to transitions and transitions to places. An enabled transition removes one token from each of the input places, and adds one token to each of its output places. This is called the firing rule. The PN graph also has an initial state called the initial marking M_0. Formally, a PN is a 5 tuple, $PN = \{P, T, F, W, M_0\}$ where, $P = \{p_1, p_2, \ldots, p_m\}$is a finite set of places, $T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions, F is a set of arcs such that $F \in (P \times T) \cup (T \times P)$, W is a weight function $W: F \rightarrow \{1, 2, \ldots\}$, M_0 is the initial marking $M_0: P \rightarrow \{1,2,3,\ldots\}$ is the initial marking. As stated earlier, the behavioral properties of the target system can be analyzed using the properties like reachability, boundedness, liveness, coverability, persistence, reversibility, fairness etc.

3.1 Components Wise Mapping from Conceptual Framework to Petri Net

In the proposed conceptual framework, agent definition has various components namely events, constraints, resources, properties, knowledge and services. All these are the individual items which together make AgOS successful to achieve the pre specified goal. In this sub section we have mapped the different components of agent definition in PN components called places and transitions.

A place P in PN comprises of set of tokens T_k belongs to constraints, resources, properties, knowledge and services of any agent. Formally, $P \rightarrow T_k$ where, $T_k \in C \cup K \cup PR \cup K \cup S$. All the events of any agent will be mapped as transitions T of a PN. Formally, $T \rightarrow E$.

The graphical notation of place and transition are represented as usual notation of PN and those are *Circle* and *Bar* respectively.

Also it is important to note that, in a PN of AgOS, due to firing of any transition *T*, all the sub components of the output place *P* will be affected simultaneously. Hence for any place in resulted PN one can set the mark as *0* or *1*.

3.2 Generic PN Representation of Conceptual Framework

As discussed earlier, an AgOS is event driven system, where agent is an autonomous entity. Irrespective of any environment, any agent in AgOS will have certain generic set of services which will be used in response to a generic set of events. Further, we have also proposed the mapping rules for the components of any agent to resultant PN for the system. These facts will result the formation of *Generic PN* for the analysis of agent's dynamic behavior.

A generic set of events associated with any agent can be *Initiate* (e_1), *Search for knowledge* (e_2), *service provided* (e_3), *Interrupt from the environment or actors* (e_4), *Activity resume* (e_5), *Activity cancel* (e_6). Further these events can be mapped into the transitions of Generic PN will be t_1, t_2, t_3, t_4, t_5 and t_6 respectively. Similarly a generic set of services can be

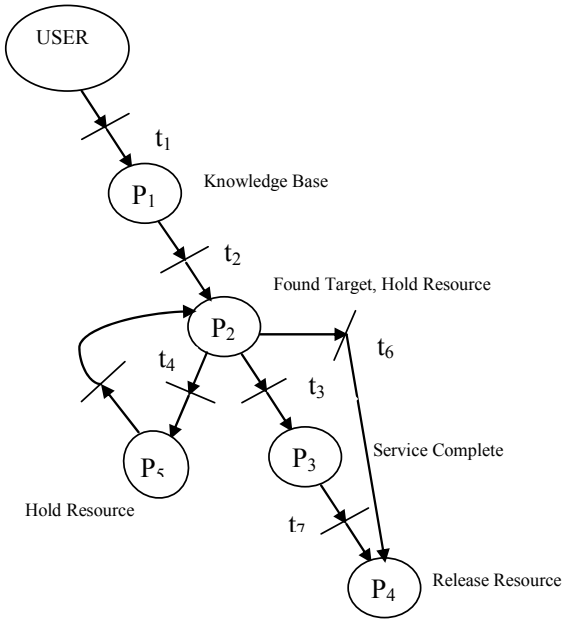


Figure 2: Generic PN Representation of Conceptual Framework

performed by any generic agent and those are *Initiate* (s_1), *Handle resources* (s_2), *Handle knowledge* (s_3), *Handle constraints* (s_4), *Handle properties* (s_5) and *Goal completed* (s_6). Those can be mapped into p_1, p_2, p_3, p_4, p_5 and p_6 respectively. The Generic PN with the specified generic set of transitions and places has been shown in Figure 2.

4 Behavioral Analysis of Conceptual Agent

The behavioral aspect of an AgOS can be studied once the resulted PN can be developed for that system. For the purpose first we need to map the components of the conceptual agents of such system into the well defined places and transitions. The mapping rules for that have been discussed in section 3. On next the several properties of the resulted PN can be studied to analyze the crucial behavioral features of the AgOS in respect to the target environment. The analysis can be performed using the incidence matrix and reachability graph of the resulted PN.

For the example described in the section 2.1, the places and transitions have been summarized in Table 1 and Table 2 respectively. The proposed mapping rules have been used to devise the said places and transitions.

Using the Table 1 and Table 2, we can draw the required PN for the system as shown in Figure 3. The process starts from a place p_0 which is the user initiate and after a transition t_1 will reach a place $\boldsymbol{p_1}$. The process continues further on and we finally arrive at the place p_{12}**.** Serially as the transitions occur the process moves on to each of the places as explained in the tables. If the process is interrupted then from place p_3 it will follow the path of places p_7, p_8, p_9 for the transition t_7, t_8, t_9, t_{10} respectively. If the process is cancelled then the path of places p_{10}, p_{11}, p_{12} will be followed for the transitions t_{11}, t_{12}, t_{13} respectively. If during the process no *IP address* is left to be processed then because of the transition t_{13} the place p_{12} is reached. If unprocessed *IP address* is still found then the process follows the path of place p_6, p_1 via transition t_{10}.

Table 1: Places for PN

Places	Components Of The Place
p_0	It consists of the Agent User.
p_1	Knowledge k_1
p_2	Service s_1; Property pr_2; resource r_2 on hold
p_3	Service s_5 s_6 s_8; Knowledge k_3 and resource r_3 updated
p_4	Service s_2; Property pr_1 and pr_4 updated
p_5	Resource r_2 ,r_3 released and r_4 restarted; Property pr_4 updated; check constraint c_3 from k_1
p_6	Check c_1 from r_4; release r_4; set property pr_3
p_7	Hold property r_1, r_2, r_3
p_8	Service s_3
p_9	Update property pr_4
p_{10}	Release r_1 r_2 r_3 r_4
p_{11}	Service s_4; Property pr_4 updated
p_{12}	Service s_7

Table 2: Transition for PN

Transitions	Details Of The Events / Transitions
t_1 for e_1	Initiate.
t_2 for e_2	User Interrupt.
t_3 for e_3	User Resume.
t_4 for e_4	User Cancel Job.
t_5 for e_5	Search.
t_6 for e_6	Target Computer Found.
t_7 for e_7	Service Init.
t_8 for e_8	Service Resume.
t_9 for e_9	Service Completed.
t_{10} for e_{10}	Service Interrupt.
t_{11} for e_{11}	Service Revoked.
t_{12} for e_{12}	Timer start.
t_{13} for e_{13}	Terminate.
t_{14} for e_{14}	No Action.

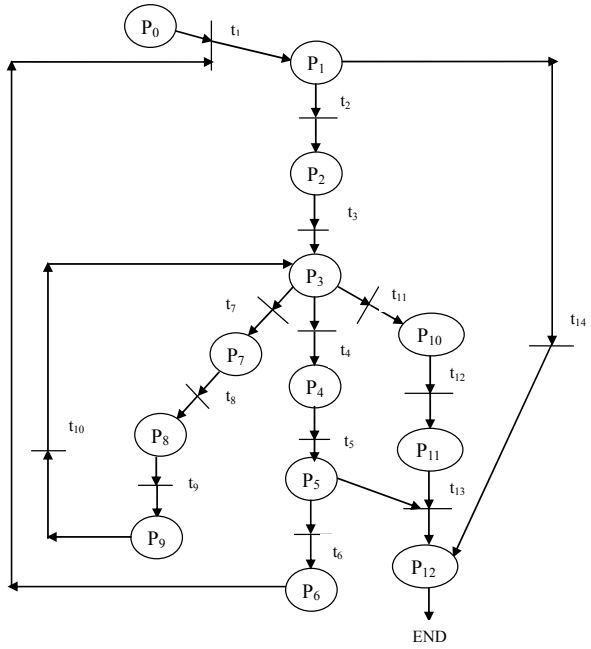


Figure 3: PN for the Agent Based Example

4.1 Incidence Matrices of PN Model

There is a pre-incidence matrix (Table 3) representing the initial state, Post-incidence matrix (Table 4) representing operational state after firing of the set of events of the agent and the combined matrix (Table 5) representing the token status at any instance after initiating the process of some agent. Each of these matrix has been formed using the row constituents p_0, p_1, …, p_{12} and the column constituents t_1, t_2, …., t_{14}.

In the resulted PN model, among the places p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12} none of them are covered and hence the net is not covered by *P* invariants. The same is the case for the transitions and the net is not covered by *T* invariants.

Table 3: Pre-incidence matrix for PN Model

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
p_0	1	1	0	0	0	0	0	0	0	0	0	0	0	1
p_1	0	0	1	0	0	0	0	0	0	0	0	0	0	0
p_2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p_3	0	0	0	1	0	0	1	0	0	0	1	0	0	0
p_4	0	0	0	0	1	0	0	0	0	0	0	0	0	0
p_5	0	0	0	0	0	1	0	0	0	0	0	0	0	0
p_6	1	0	0	0	0	0	0	0	0	0	0	0	0	0
p_7	0	0	0	0	0	0	0	1	0	0	0	0	0	0
p_8	0	0	0	0	0	0	0	0	1	0	0	0	0	0
p_9	0	0	0	0	0	0	0	0	0	1	0	0	0	0
p_{10}	0	0	0	0	0	0	0	0	0	0	0	1	0	0
p_{11}	0	0	0	0	0	0	0	0	0	0	0	0	1	0
p_{12}	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 4: Post-incidence Matrix for PN Model

	t_1	T_2	T_3	t_4	T_5	t_6	t_7	t_8	t_9	t_{10}	T_{11}	t_{12}	t_{13}	t_{14}
p_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
p_1	1	0	0	0	0	0	0	0	0	0	0	0	0	0
p_2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
p_3	0	0	1	0	0	0	0	0	0	1	0	0	0	0
p_4	0	0	0	1	0	0	0	0	0	0	0	0	0	0
p_5	0	0	0	0	1	0	0	0	0	0	0	0	0	0
p_6	0	0	0	0	0	1	0	0	0	0	0	0	0	0
p_7	0	0	0	0	0	0	1	0	0	0	0	0	0	0
p_8	0	0	0	0	0	0	0	1	0	0	0	0	0	0
p_9	0	0	0	0	0	0	0	0	1	0	0	0	0	0
p_{10}	0	0	0	0	0	0	0	0	0	0	1	0	0	0
p_{11}	0	0	0	0	0	0	0	0	0	0	0	1	0	0
p_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	1

Table 5: Combined Matrix for PN Model

	t_1	T_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}
p_0	-1	0	0	0	0	0	0	0	0	0	0	0	0	-1
p_1	1	-1	0	0	0	0	0	0	0	0	0	0	0	0
p_2	0	1	0	0	0	0	0	0	0	0	0	0	0	0
p_3	0	0	1	-1	0	0	-1	0	0	0	-1	0	0	0
p_4	0	0	0	1	-1	0	0	0	0	0	0	0	0	0
p_5	0	0	0	0	1	-1	0	0	0	0	0	0	-1	0
p_6	-1	0	0	0	0	1	0	0	0	0	0	0	0	0
p_7	0	0	0	0	0	0	1	-1	0	0	0	0	0	0
p_8	0	0	0	0	0	0	0	1	-1	0	0	0	0	0
p_9	0	0	0	0	0	0	0	0	1	-1	0	0	0	0
p_{10}	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
p_{11}	0	0	0	0	0	0	0	0	0	0	0	1	-1	0
p_{12}	0	0	0	0	0	0	0	0	0	0	0	0	1	1

4.2 Reachability Graph

The reachable place of a PN can be expressed by the reachability graph, which is a directed graph. The nodes of the graph are identified as markings of the net $R(N, M_0)$, where M_0 is the initial marking and the arcs are represented by the transitions of N. The graph is used to define a given PN N and marking M, where M belongs to $R(N)$. Each initial marking M_0 has an associated *Reachability* set. This set consists of all the markings that can be reached from M_0 through the firing of one or more transitions. In our PN model the reachability graph starts with initial marking $M_0 = [1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]^T$ and finally reach to state $M_{12} = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 1]^T$, where we conclude the agent session for the current process of shutdown. The reachability graph has been shown in Figure 4.

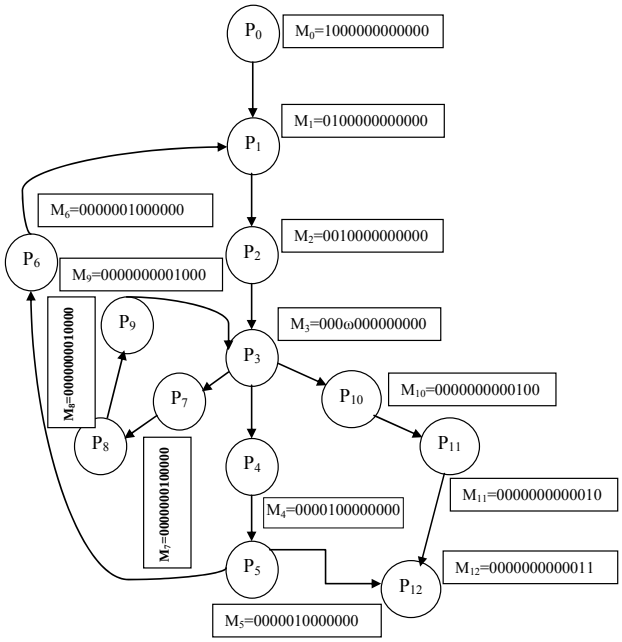


Figure 4: Reachability Graph for PN Model

As stated earlier, the mark of each place in P of PN model will be treated as 1 or 0 because all the agent related sub components of any output place $p_i \in P$ will be affected simultaneously.

4.3 Analysis of PN Model

In this sub section some of the crucial properties and behavior of the PN model of the example agent have been analyzed using the PN model and Reachability Graph presented in Figure 4 and Figure 5 respectively.

(a) Safeness: Any place of a Reachability graph is declared safe, if the number of tokens at that place is either *0* or *1*. In our PN model, the graph clearly shows that any of the places within p_0 to p_{12} represents a combination of *0* (no token) and *1* (token), which implies that if the firing occurs there will be a token at the position bit otherwise no token. Thus it shows each of the places has a maximum token count *1* or *0* and is declare safe. Also as all the places in the net are safe, the net as a whole can be declared safe.

(b) Boundedness: The boundedness is a generalized property of safeness. The limitation of token numbers in a place is restricted to *1* in case it is safe. It may enhance to some integer *i*, where *i* is known before hand for a place or we call it as a constraint to check the overflow condition at any stage calculated once the agent process start. When there is no overflow at any place, then the design guarantees the boundedness of the model. In our case there is no deadlock and at any stage within p_0 to p_{12} and hence it is bounded.

(c) Reachable: Reachability is a fundamental basis for studying the dynamic properties of any system. The firing of an enabled transition will change the token distribution

(markings) in a net according to the transition rules. A sequence of firings will result in a sequence of markings. A marking M_n is reachable from some marking M_0, if there exist a sequence of transitions that transforms M_0 to M_n. In our example all the markings are reachable starting from any marking in the net and hence reachability exists. This guarantees that the PN model for the AgOS will meet the pre-specified goal.

(d) Liveness: The liveness property of a PN is used to show continuous operation of the net model and ensure that the system will not get into a deadlock state as the process of interrupt or cancel or initiate needs to perform some transitions. If any marking exists in the graph such that no transitions are enabled from that marking, then that marking represents a deadlocked state, and the PN lacks the liveness property. Otherwise it is declared live. In our case there is no such deadlocked state or marking present in the net due to a series of events for the specified agent. Hence the net is live. Also in the example, if we have *k* computers to be shutdown then it means that transition *t* can be fired at least *k* number of times in some firing sequence. Hence the PN model is *L2* live.

(e) Conservativeness: Conservation property of a PN model checks the number of tokens remains constant before and after the execution. The process is to count the sum of all tokens at their initial markings and again after the execution. If all the markings in the reachability graph have the same sum of tokens then the Petri net is declared to be strictly conservative. The PN model of AgOS example is also strictly conservative.

5 Conclusion

In this paper, a methodology has been proposed to analyze the dynamic behavior of AgOS. Any such system comprises of autonomous entity called agent. They are highly dynamic in nature in terms of its interactions with the environments, handling of environmental resources, activities to achieve the pre specified goal and acquisition of knowledge. Petri net is best suitable tools to model and analyze such system behavior. To conceptualize the agent based system one need to follow entirely new paradigm than the object oriented paradigm. In this paper, firstly, we have proposed a conceptual level framework for agent as well as for such system to represent AgOS in simpler form and which can comply with the crucial features of such system. On next, to model the dynamic behavior of agent, we have used classical Petri Net as tool. A set of mapping rules also have been proposed for presenting the elements of conceptual framework for agent into the Petri net components. It also resulted a Generic Petri Net model for the agent oriented system. Finally we have used the Petri net model and its reachability graph to analyze the dynamic features of agent oriented system.

The model works fine for the system composed of simple agents. The proposed methodology also is useful for the analysis of external and internal behavior of agents. But the methodology will become less expressive for large system comprised of multiple agents and with complex agent level interactions. Future work includes, the development of a mechanism for more expressive behavioral analysis model of agent oriented system using High Level Petri Net by extending the proposed model.

6 References

[1] M. Wooldridge, P. Ciancarini, *"Agent-Oriented Software Engineering: The State of the Art"*, Book Title: Agent-Oriented Software Engineering, Springer – Verlag Lecture Notes in AI, Vol. 1957, pp 1-28, January 2001.

[2] Tadao Murata, *"Petri Nets: Properties, Analysis and Applications"*, Proceedings of the IEEE, Vol. 77, No. 4, pp 541 – 580, April 1989.

[3] F. Zambonelli, A. Omicini, *"Challenges and Research Directions in Agent-Oriented Software Engineering"*, Jnl. of Autonomous Agents and Multi-Agent Systems, Vol. 9, pp 253–283, 2004.

[4] B. Bauer, J. P. Müller, J. Odell, *"Agent UML: A Formalism for Specifying Multiagent Software Systems"*, International Journal of Software Engineering and Knowledge Engineering, Vol 11, No. 3, pp.1-24, 2001.

[5] P. Gruer, V. Hilaire, A. Koukam and K. Cetnarowicz, *"A Formal Framework for Multi-Agent Systems Analysis and Design"*, Journal of Expert Systems with Applications, Vol. 23, No. 4, pp. 349–355, 2002.

[6] P. K. Biswas, *"Towards an agent-oriented approach to conceptualization"*, Journal of Applied Soft Computing, Vol. 8, No. 1, pp 127-139, January 2008.

[7] F. Zambonelli, N. R. Jennings, M. Wooldridge, *"Developing Multiagent Systems: The Gaia Methodology"*, ACM Trans. on Software Engineering and Methodology, Vol. 12, No. 3, pp 317 – 370, 2003.

[8] S. A. Deloach, M. F. Wood, C. H. Sparkman, *"Multiagent Systems Engineering"*, International Journal of Software Engineering and Knowledge Engineering, Vol. 11, No. 3, pp 231 – 258, 2001.

[9] F. Giunchiglia, J. Mylopoulos, A. Perini, *"The Tropos Software Development Methodology: Processes, Models and Diagrams"*, 3rd Intl. Conference on Agent-Oriented Software Engineering (AOSE'02), pp 162-173, 2003.

[10] Jürgen Lind, *"Issues in Agent-Oriented Software Engineering"*, Transaction on Agent-Oriented Software Engineering, Springer-Verlag pp 45-58, June 2009 .

[11] B. Bauer, J. P. Muller, J. Odell, *"Agent Uml: A formalism for specifying Multiagent Software Systems"*, International Journal of Software Engineering and Knowledge Engineering, Vol. 11, No. 3, pp 1 – 24, 2001.

[12] B. Marzougui, K. Hassine, K. Barkaoui, *"A New Formalism for Modeling a Multi Agent Systems: Agent Petri Nets"*, Journal of Software Engineering and Applications, Vol. 3, No. 12, pp 1118-1124, 2010.

Graph Semantic Based Conceptual Model of Semi-structured Data: An Object Oriented Approach

Anirban Sarkar[1], Sesa Singha Roy[2]
[1]Department of Computer Applications, National Institute of Technology, Durgapur, India
[2]Department of Computer Science & Engineering, University Institute of Technology, Burdwan, India
{sarkar.anirban@gmail.com, sesa.sroy.02nit@gmail.com}

Abstract - *This paper has proposed a Graph – semantic based conceptual data model (GOOSSDM) for Semi-structured data, to conceptualize the different facets of such system in object oriented paradigm. The proposed model facilitates modeling of irregular, heterogeneous, hierarchical and non-hierarchical semi-structured data at the conceptual level. The GOOSSDM is also able to represent the mixed content in semi-structured data. Moreover, the proposed approach is capable to model XML document at conceptual level with the facility of document-centric design, ordering and disjunction characteristic. A rule based transformation of GOOSSDM schema into the equivalent XML Schema Definition (XSD) also has been proposed in this paper.*

Keywords: Semi-structured Data, XML, XSD, Conceptual Modeling, Semi-structured Data Modeling, XML Modeling.

1 Introduction

The increasingly large amount of data processing on the web based applications has led a crucial role of semi-structured database system. Semi-structured data though is organized in semantic entity but does not strictly conform the formal structure to strict types. Rather it posses irregular and partial organization [1]. Further semi-structured data evolve rapidly and thus the schema for such data is large, dynamic, and also is not considered strict participation of instances.

The eXtensible Markup Language (XML) using Document Type Definition (DTD) or XML Schema Definition (XSD) is increasingly finding acceptance as a standard for storing and exchanging structured and semi-structured information over internet [12]. However, the XML schemas provide the logical representation of the semi-structured data and it is hard to realize the semantic characters of such data. Thus it is important to devise a conceptual representation of semi-structured data for designing the information system based on such data more effectively. A conceptual model of semi-structured data deals with high level representation of the candidate application domain in order to capture the user ideas using rich set of semantic constructs and interrelationship thereof. The conceptual design of such system further can be implemented in XML based logical model.

To adopt the rapidly data evolving characteristics, the conceptual model of semi-structured data must support several properties like, representation of irregular and heterogeneous structure, hierarchical relations along with the non-hierarchical relationship types, cardinality, n-array relation, ordering, representation of mixed content etc. [13]. In early years, Object Exchange Model has been proposed to model semi-structured data [2], where data are represented using directed labeled graph. The schema information is maintained in the labels of the graph and the data instances are represented using nodes. However, the separation of the structural semantic and content of the schema is not possible in this approach. In recent past, several researches have been made on conceptual modeling of semi-structured data as well as XML. Many of these approaches [3, 4, 5, 6, 7, 8] have been extended the Entity Relationship (ER) model to accommodate the facet of semi-structured data at conceptual level. The major drawbacks of these proposals are in representation of hierarchical structure of semi-structured data. Moreover, only two ER based proposals [7, 8] support the representation of mixed content in conceptual schema. On the other hand, ORA-SS [11] proposed to realize the semi-structured data at conceptual level starting from its hierarchical structure. But the approach does not support directly the representation of no-hierarchical relationships and mixed content in conceptual level semi-structured data model.

Very few attempts have been made to model the semi-structured data using Object Oriented (OO) paradigm. ORA-SS[11] support the object oriented characteristic partially. The approaches proposed in [9, 10, 12] are based on UML and have extended the UML stereotype definitions and notations. However, the UML and extensions to UML represent software elements using a set of language elements with fixed implementation semantics (e.g. methods, classes). Henceforth, the proposed approaches using extension of UML, in general, are logically inclined towards implementation of semi-structured database system. This may not reflect the facet of such system with high level of abstraction to the user and thus cannot be considered as semantically rich conceptual level model.

In this paper, we have proposed a graph semantic based conceptual model for semi-structured data called Graph Object Oriented Semi-Structured Data Model (GOOSSDM). The model is comprehensively based on object oriented

paradigm. Among others, the proposed model supports the representation of hierarchical structure along with non-hierarchical relationships, mixed content, ordering, participation constraints etc. The proposed GOOSSDM reveals a set of concepts to the conceptual level design phase of semi-structured database system, which are understandable to the users, independent of implementation issues and provide a set of graphical constructs to facilitate the designers of such system. The schema in GOOSSDM is organized in layered approach to provide different level of abstraction to the users and designers. In this approach a rule based transformation mechanism also has been proposed to represent the equivalent XML Schema Definitions (XSD) from GOOSSDM schemata.

The work is motivated from Graph Data Model (GDM) [14] and Graph Object Oriented Multidimensional Data Model (GOOMD) [15], which supports structural abstraction of OnLine Transaction Processing database and OnLine Analytical Processing (OLAP) databases respectively.

2 GOOSSDM: The Proposed Model

The GOOSSDM extends the object oriented paradigm to model of semi-structured data. It allows the entire semi-structured database to be viewed as a Graph (V, E) in layered organization. At the lowest layer, each vertex represents an occurrence of an attribute or a data item, e.g. name, day, city etc. A set of vertices semantically related is grouped together to construct an Elementary Semantic Group (ESG). So an ESG is a set of all possible instances for a particular attribute or data item. On next, several related ESGs are group together to form a Contextual Semantic Group (CSG). Even the related ESGs with non-strict participations or loosely related ESGs are also constituent of related CSG – the constructs of related data items or attributes to represent one semi-structured entity or object. The edges within CSG are to represent the containment relation between different ESG in the said CSG. The most inner layer of CSG is the construct of highest level of abstraction in semi-structured schema formation. This layered structure may be further organized by combination of one or more CSGs as well as ESGs to represent next upper level layers and to achieve further lower level abstraction or higher level in the semi-structure data schema hierarchy. From the topmost layer the entire database appears to be a graph with CSGs as vertices and edges between CSGs as the association amongst them. The CSGs of topmost layer will act as roots of semi-structured data model schemata.

2.1 Modeling Constructs in GOOSSDM

Since from the topmost layer, a set of vertices V is decided on the basis of level of data abstraction whereas the set of edges E is decided on basis of the association between different semantic groups. The basic components for the model are as follows,

(a) A set of *t* distinct *attributes* $A = \{a_1, a_2, \ldots, a_t\}$ where, each a_i is an attribute or a data item semantically distinct.

(b) Elementary Semantic Group (ESG): An elementary semantic group is an encapsulation of all possible instances or occurrences of an attribute, that can be expressed as graph ESG (V, E), where the set of edges E is a null set ∅ and the set of vertices *V* represent the set of all possible instances of an attribute $x_i \in A$. ESG is a construct to realize the elementary property, parameter, kind etc. of some related concern. Henceforth there will be set of *t* ESGs and can be represented as $E_G = \{ESG_1, ESG_2, \ldots, ESG_t\}$. The graphical notation for the any ESG is *Circle*.

(c) Contextual Semantic Group (CSG): A lowest layer contextual semantic group is an encapsulation of set of ESGs or references of one or more related ESGs to represent the context of one entity of semi-structured data. Let, the set of *n* CSGs can be represented as $C_G = \{CSG_1, CSG_2, \ldots, CSG_n\}$. Then any lowest layer $CSG_i \subseteq C_G$ can be represented as a graph (V_{Ci}, E_{Ci}) where vertices $V_{Ci} \subseteq E_G$ and the set of edges E_{Ci} represents the association amongst the vertices. For any CSG, it is also possible to designate one or more encapsulated ESGs as *determinant vertex* which may determine an unordered or ordered set of instances of constituent ESGs or CSGs. The graphical notation for any CSG is *square* and determinant vertex is *Solid Circle*.

Composition of multiple CSGs can be realized in two ways. *Firstly*, the simple *Association* (Discussed in subsection II.B) may be drawn between two or more associated CSGs either of same layer or of adjacent layers to represent the non-hierarchical and hierarchical data structure respectively. The associated CSG will be connected using *Association Connector*. *Secondly*, lower layer CSGs may maintain an *Inheritance* or *Containment* relationship (Discussed in subsection 2.B) with the adjacent upper layer CSG to represent the different level of abstraction.

(d) ***Annotation*****:** Annotation is a specialized form of CSG and can be expressed as *G(V, E)*, where $|V| = 1$ and $E = \varnothing$. Annotation can contain only text content as tagged value. Annotation can be containment in or associated with any other CSG. Further the cardinality constraint for Annotation construct is always *1:1* and ordering option can be *1* or *0*, where *1* means content will be in orderly form with other constituent ESG and *0* means text content can be mixed with other constituent ESGs. The annotation construct will realize the document-centric semi-structured data possibly with mixed content. This concept is extremely important for mapping semi-structured data model in XML. Graphically Annotation can be expressed using *Square with Folded Corner*.

The summary of GOOSSDM constructs and their graphical notations have been given in Table 1.

Table 1: Summary of GOOSSDM constructs and their graphical notations

GOOSSDM Constructs	Description	Graphical Notation
ESG	Elementary Semantic Group	
Determinant ESG	Determinant vertex of any CSG which will determine the other member vertices in the CSG	
CSG	Contextual Semantic Group	
Annotation	Specialized form of CSG. Contain only Text Content.	
Association Connector	Connect multiple associated CSGs	

2.2 Relationship Types in GOOSSDM

The proposed GOOSSDM provides a graph structure to represent semi-structured data. The edges of the graph represent relationships between or within the constructs of the model. In the proposed model, *four* types of edges have been used to represent different relationships. The type of edges and their corresponding meanings are as follows,

(a) Containment: Containments are defined between encapsulated ESGs including determinant ESG and parent CSG, or between two constituent CSGs and parent CSG, or between CSG and referential constructs. The Containment relationship is constrained by the parameters tuple $<p, \theta>$, where p determines the participation of instances in containment and θ determines the ordering option of constituent ESGs or CSGs. With any CSG, this represents a bijective mapping between determinant ESG and other ESGs or composed CSG with participation constraint. Graphically association can be expressed using *Solid Directed Edge* from the constituent constructs to its parent labeled by constraint specifications. The possible values for *p* are,

(i) 1:1 – Represents ESG with total participation in the relationship. This is default value of *p.*
(ii) 0:1 – Represents ESG with optional one instantiation in the relationship.
(iii) *1:M* – Represents ESG with mandatory multiple instantiation in the relationship.
(iv) *0:M* – Represents ESG with optional multiple instantiation in the relationship.
(v) *0:X* – Represents ESGs with optional exclusive instantiation in the relationship.
(vi) *1:X* – Represents ESG with mandatory exclusive instantiation in the relationship.

The possible values for θ are as follows,
(i) *1* – Represents that for any CSG, the constituent ESGs and CSGs are ordered and order must be maintained from left to right in the list of ESGs with θ value 1.
(ii) *0* – This is default value of θ and represents that for any CSG, the constituent ESGs and CSGs are not ordered.

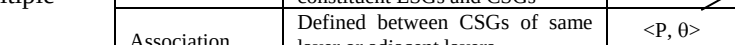
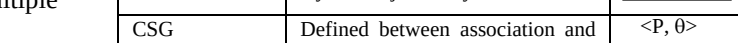
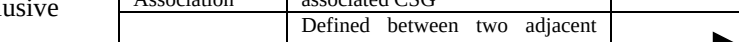
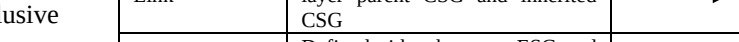
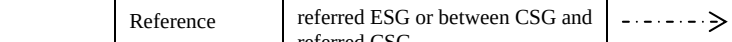
(b) Association: Associations are defined between related CSGs of same layer of adjacent layers. The Association relationship is constrained by the parameters tuple $<P, \theta>$, where P determines the cardinality of Association and θ determines the ordering option of associated CSGs. Graphically association can be expressed using *Solid Undirected Edge.* Any CSG wish to participate in association will be connected with association relationship. On next, multiple associated CSGs will be connected through *Association Connector*. For semi-structured it is sometime important to have specific context of some association. Such context can be represented using *Associated CSG* defined on *Association Connector*. Association Connector facilitates the n – array relationship. Graphically association can be expressed using *Solid Undirected Edge,* Association connector can be expressed using *Solid Diamond* and Associated CSG can be connected with Association Connector using *Dotted Undirected Edges* with Participation constraint specifications. The values for P can be *1:1* or *0:1* or *1:N* or *0:N* or *0:X* or *1:X* with corresponding meaning and possible values for θ can be *1* or *0* with corresponding meaning.

(c) Link: Links are used to represent the inheritance relationships between two CSGs. Graphically link can be expressed using *Solid Directed Edge with Bold Head* from the generalized CSG to the specialized one.

(d) Reference: In semi-structured data model, it is important to represent the symmetric relationship between ESGs or CSGs. Reference can be used to model such concepts. Reference relations are defined either between ESG and referred ESG or between CSG and referred CSG. Graphically reference can be expressed using *Dotted Directed Edge.*

The summary of GOOSSDM relationship types and their graphical notations have been given in Table 2.

Table 2: Summary of GOOSSDM relations types and their graphical notations

GOOSSDM Relationships	Description	Graphical Notation
Containment	Defined between Parent CSG and constituent ESGs and CSGs	<p, θ> 
Association	Defined between CSGs of same layer or adjacent layers.	<P, θ> 
CSG Association	Defined between association and associated CSG	<P, θ> 
Link	Defined between two adjacent layer parent CSG and inherited CSG	
Reference	Defined either between ESG and referred ESG or between CSG and referred CSG.	

3 Transformation of GOOSSD into XSD

In general, the proposed GOOSSDM can be useful to realize the semi-structured data schema at conceptual level. The logical structure of such schema can be represented using

the artifacts of XSD. Moreover, XSD is currently the de facto standard for describing XML documents. An XSD schema itself can be considered as an XML document. Elements are the main building block of any XML document. They contain the data and determine the elementary structures within the document. Other wise, XSD also may contain sub-element, attributes, complex types, and simple types. XSD schema elements exhibit hierarchical structure with single root element.

A systematic rule based transformation of GOOSD schemata to XSD is essential to express the semi-structured data at logical level more effectively. For the purpose, a set of rules have been proposed to generate the equivalent XSD from the semantic constructs and relationship types of a given GOOSSDM schemata. Based on the concepts of GOOSSDM constructs and relationship types the transformation rules are as follows,

Rule1: An ESG will be expressed as an *element* in XSD. For example, ESG_{City} can be defined on attribute *City* to realize Customer city. Any DESG construct must be expressed with typed *ID* in XSD. The equivalent representation in XSD can be as given in Figure 1.

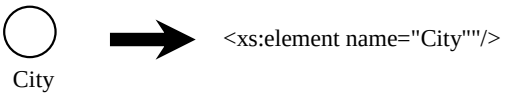


Figure 1: Representation of ESG.

Rule 2: A CSG will be expressed as a *complexType* in XSD. For example, $CSG_{Customer}$ can be defined to realize the detail of Customer. The equivalent representation in XSD can be as given in Figure 2.

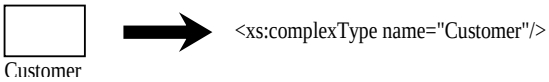


Figure 2: Representation of CSG.

Rule 3: Any Annotation construct will be expressed as a *complexType* in XSD with suitable *mixed* value. On containment to other CSG, if θ value is 0 then it will be treated as mixed content in the resulted XML document. Otherwise if θ value is 1 then it will be treated as annotation text in resultant XML document in orderly form.

Rule 4: A Reference of ESG and CSG will be expressed as a *complexType* in XSD.

Rule 5: CSGs of topmost layer will be treated as root in XSD declaration.

Rule 6: Any lowest layer CSG with containment of some ESGs will be expressed as a *complexType* with *elements* declaration in XSD. Further the participation constraint (*p value* in GOOSSDM concept) can be expressed using *minOccurs* and *maxOccurs* attribute in XSD. The ordering constraint (θ *value* in GOOSSDM concept) can be expressed using compositor type of XSD. If θ *value* is 1 then compositor type will be *sequence* otherwise *all*. For ordered set ESGs, the order will be maintained from *left to right*. If any subset of ESGs contain the *p value X:1* then those ESGs will be composite using *choice* compositor type in XSD.

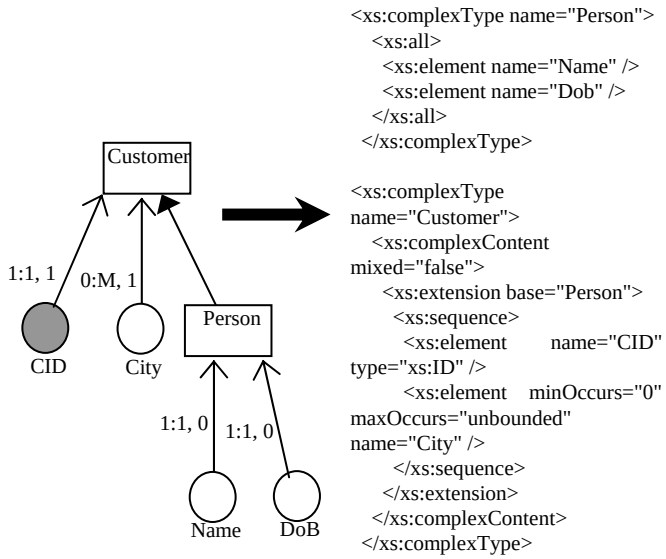


Figure 3: Representation of Link Relationship

Rule 7: Any upper layer CSG with containment of ESGs, reference of ESGs and adjacent lower layer CSGs will be expressed as a *complexType* in XSD.

Rule 8: Any upper layer CSG with *Link* relationship with adjacent lower layer CSGs will be expressed as a *complexType* with inheritance in XSD. Upper layer CSG will be the child of lower layer CSG. An example of XSD representation of upper layer CSG with inheritance with adjacent lower layer CSG has been shown in Figure 3.

Rule 9: Any upper layer CSG with *Association* relationship with adjacent lower layer CSGs will be expressed as a *complexType* with nesting in XSD. Upper layer CSG will be treated as root element.

Rule 10: Association relationship between any two CSGs in the same layer will be expressed as a *complexType* with nesting in XSD. Rightmost CSG will be treated as the root element and on next nesting should be done in *right to left* order of the CSG in the same layer.

Rule 11: N – array *Association* relationship within a set of CSGs spread over several layer will be expressed as a *complexType* with nesting in XSD. Topmost layer CSG will be treated as the root element in XSD. Then, in the adjacent lower layer the rightmost CSG should be treateda as nested element within the root element. Further the nesting should be done in right to left order of the CSG in the same layer and on next moving on the adjacent lower layers.

Rule 12: With several *Association* relationship (composition of n – array and simple relationship) within a set of CSGs spread over several layer will be expressed as a *complexType* with nesting in XSD. Topmost layer CSG will be treated as the root element in XSD. Then, if available, the directly associated CSGs in each adjacent lower layer will be nested

till it reaches to the lowermost layer of available associated CSG. On next, the CSGs of adjacent lower layer of the root element will be nested from *right to left* order in the same layer along with the nesting of directly associated CSGs (if available) in each corresponding adjacent layers.

4 Case Study

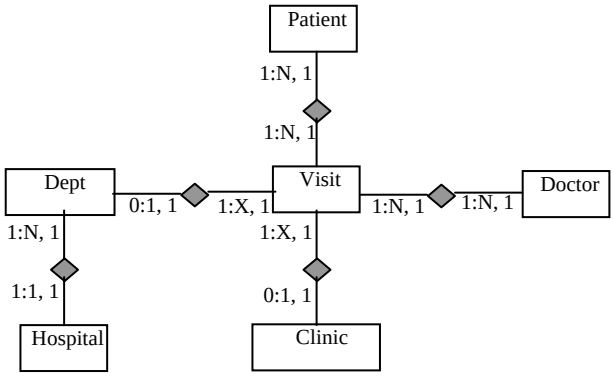
Let consider an example of *Patient and Doctor* where a *Patient* can visit to a *Doctor* either at *Hospital Department* or at *Clinic*. Any Patient can visit several times to different doctor [7]. Figure 4 shows an irregularly structured XML representation of visit records of two patients. *Patient 1* visited twice to two different *Doctors*, one at *Hospital Department* and another at *Clinic. Patient 2* visited once to one *Doctor* common to *Patient 1* but at different *Clinic*. All though in the document, the Date of Visit, Doctor and option of Hospital Department and Clinic are in order. The XML document of Figure 4 represents the semi-structured data. The suitable GOOSSDM schemata for such data and its equivalent XSD have been shown in Figure 5. The equivalent XSD of GOOSSDM schemata of Figure 5 can be generated using the rules described in Section 3.

```
<Patient>
 <name>Patient 1</name>
 <Visit>
   <Date>10-JAN-2009</Date>
   <Doctor>
      <RegID>1234</RegID>
      <DName>Dr. P. Roy</DName>
   </Doctor>
   <Dept><DID>1</DID><DeptName>General</DeptName>
         <Hospital><Name>Hospital A</Name>  </Hospital>
   </Dept>

   <Date>15-MAR-2010</Date>
   <Doctor>
      <RegID>4321</RegID>
      <DName>Dr. T. De</DName>
   </Doctor>
   <Clinic><Name>Clinic B</Name></Clinic>
 </Visit>

 <name>Patient 2</name>
 <Visit>
   <Date>12-SEPT-2009</Date>
   <Doctor>
      <RegID>4321</RegID>
      <DName>Dr. T. De</DName>
   </Doctor>
   <Clinic><Name>Clinic D</Name></Clinic>
 </Visit>
</Patient>
```

Figure 4: Irregular Structure in Visit Records XML.




```
<xs:element name="Patient">
  <xs:complexType>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="name" />
    <xs:element maxOccurs="unbounded" name="Visit">
     <xs:complexType>
      <xs:sequence maxOccurs="unbounded">
       <xs:element name="Date" />
       <xs:element name="Doctor">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="RegID" />
          <xs:element name="DName" />
         </xs:sequence>
        </xs:complexType>
       </xs:element>
       <xs:choice>
        <xs:element minOccurs="0" name="Dept">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="DID" type="xs:ID" />
           <xs:element name="DeptName" />
           <xs:element name="Hospital">
            <xs:complexType>
             <xs:sequence>
              <xs:element name="Name" />
             </xs:sequence>
            </xs:complexType>
           </xs:element>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element minOccurs="0" name="Clinic">
         <xs:complexType>
          <xs:sequence>
           <xs:element name="Name" />
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:choice>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 5: Corresponding GOOSSDM schemata and Equivalent XSD of Figure 4.

5 Feature of GOOSSDM

The proposed GOOSSDM is an extension of comprehensive object oriented model for Semi-structured Database System and which can be viewed as a Graph (V, E) in layered organization. Apart from that, one of the major advantages of the model is that it defines each level of structural detail on the constructs which are independent of the implementation issues. Moreover, the graph structure maintains the referential integrity inherently. The features of the proposed model are as follows,

(a) Explicit Separation of structure and Content: The model provides a unique design framework to specify the design for the semi-structured database system using semantic definitions of different levels (from elementary to composite) of data structure through graph. The model reveals a set of structures like ESG, CSG, Annotation, Association Connector etc. along with a set of relationships like Containment, Association, Link, Reference etc. between the structures, which are not instance based or value based. So, the nature of contents that corresponded with the instances and the functional constraint on the instances has been separated from the system's structural descriptions.

(b) Abstraction: In the proposed GOOSSDM, the concept of layers deploys the abstraction in semi-structured data schema. The upper layer views will hide the detail structural complexity from the users. Such a representation is highly flexible for the user to understand the basic structure of semi-structure database system and to formulate the alternative design options.

(c) Reuse Potential: The proposed model is based on Object oriented paradigm. It is supported with inheritance mechanism using the *Link* relationship. Henceforth, there is no binding in the model to reuse some CSG constructs of any layer. On reuse of CSG, the specialized CSG must be shown in adjacent upper layer of the parent CSG. Moreover, lowest layer ESG or lower layer can be shared and reused with different CSGs of the upper layers using *Containment* relationship.

(d) Disjunction Characteristic: The instances of semi-structured data schema are likely to be less homogeneous than structured data. Disjunction relationships facilitate the possibility of non-homogeneous instances. The proposed GOOSSDM supports disjunction relationship using the participation constraint attribute *p* or *P* (by setting *p or P* value either *0:X* or *1:X*). The Containment relationships between constituent ESGs or CSGs with the parent CSG can be disjunctive or Association relationships between two or more CSGs can be disjunctive. The example in Figure 5 explains such disjunctions.

(e) Hierarchical and Non-hierarchical Structure: The proposed model explicitly supports both hierarchical and non-hierarchical representation in semi-structure data modeling at conceptual level. Associated CSGs of different or same layers form the hierarchical or non-hierarchical structure in semi-structured data model. At the logical level modeling of semi-structured data using XSD supports only hierarchical structure. For the purpose, we have proposed set of rules to transformed more generous conceptual level schema to hierarchical logical schema.

(f) Ordering: Ordering is one important concept in modeling of semi-structured data. One or more attributes or relationships in semi-structured data schema can be ordered. Our proposed model supports ordering in two ways using the relationship ordering constraint attribute θ. Firstly, the ordering may be enforced between parent CSG and any set of constituent ESGs and CSGs by specifying the θ value on containment relationship. Secondly, the ordering can be enforced on the any set of Association relationships within CSG.

(g) Irregular and Heterogeneous structure: By characteristic the semi-structured data is irregular and heterogeneous. The proposed GOOSSDM supports disjunction characteristic, ordering and representation of both hierarchical and non-hierarchical structure in the same schema. With all these facets, the proposed model can efficiently model the irregular and heterogeneous semi-structured data. Modeling of irregular structure using GOOSSDM has been shown in Figure 5.

(h) Participation constraint: Instances participations in the semi-structured data schema are not followed strictly. Participations of instances can be optional or mandatory or even exclusive for such schema. This can affect the participation of constituent ESGs and CSGs in the parent CSG or may affect the participation of CSGs in some association relationship either of simply type or n-array type. All these participation constraint can be modeled in proposed GOOSSDM by specifying the value for participation constraint attribute *p* or *P*.

(i) Document-centric and Mixed Content: In real world, document texts are mixed with semi-structured data. The feature is more important and frequent in XML documents. Thus it is an essence that, the conceptual model for semi-structured data must support modeling of such feature. In the proposed model, the *Annotation* construct facilitates to model document centric design of semi-structured data at conceptual level. Moreover, the modeling of the *Annotation* construct in the GOOSDM schema allows the instances of CSG and ESG to be mixed with the text content. The presence of this construct along with the other defined constructs and relationships, the proposed GOOSSDM is also capable to model XML document at conceptual level.

6 Conclusion

In this paper, a model has been introduced for the conceptual level design of semi-structured data using graph based semantics. This is a comprehensive object oriented

conceptual model and the entire semi-structure database can be viewed as a Graph (V, E) in layered organization. The graph based semantics in GOOSSDM model extracts the positive features of both Object and Relational data models and also it maintains the referential integrity inherently.

The proposed GOOSSDM contains detailed set of semantically enriched constructs and relationships those are necessary to specify the facets of semi-structured database system at conceptual level. Moreover, a set of rules also have been proposed to systematically transform any GOOSSDM schema to its equivalent XSD structure. The expressive powers of the set of transformation rules have been illustrated with suitable example. We have also demonstrated the features and the capabilities of the proposed model by providing case study. Further the layered organization of the model facilitates to view the semi-structured data schema from different level of abstraction.

Moreover, the proposed model also facilitates the designer to provide alternative design of same schema by changing the ordering scheme, which in result can be transformed in different XSDs with different nesting patterns. The proposed GOOSSDM is enriched with set of constructs, variety of relationship types and rich set of participation constraints semantics. It provides better understandability to the users and high flexibility to the designers for creation and / or modification of semi-structured data as well as XML document at conceptual level. The proposed approach is also independent from any implementation issues.

It is also important to note that with the concept of Annotation construct the proposed GOOSSDM facilitate the document – centric design of semi-structured data at conceptual level. Also the proposed model supports irregular, heterogeneous, hierarchical and no-hierarchical structure in data. Moreover, the set of proposed rules are capable to transform systematically the GOOSSDM schema into hierarchical XSD schema. Due to these features, the proposed approach is also capable to design XML document at conceptual level.

Future studies will concentrate on developing tools support for automatic generation of XSD schema and skeleton XML document directly from the conceptual level semi-structured data model GOOSSDM and its graphical notations. A graphical query language for the proposed approach also will be a key future interest.

7 References

[1] Abiteboul S., Buneman P., Suciu D., *"Data on the Web: From Relations to Semistructured Data and XML"*, Morgan Kaufman, 1999.

[2] McHugh J., Abiteboul S., Goldman R., Quass D., Widom J., *"Lore: a database management system for semistructured data"*, Vol. 26, Issue 3, PP: 54 - 66, 1997.

[3] Badia, A., *"Conceptual modeling for semistructured data"*, 3rd International Conference on Web Information Systems Engineering, PP: 170 – 177, 2002.

[4] Mani M., *"EReX: A Conceptual Model for XML"*, 2nd International XML Database Symposium, PP 128-142, 2004.

[5] Psaila G., *"ERX: A Conceptual Model for XML Documents"*, ACM Symposium on Applied Computing, PP: 898 – 903, 2000.

[6] Sengupta A., Mohan S., Doshi R., *"XER - Extensible Entity Relationship Modeling"*, The XML 2003 Conference, PP: 140 – 154, 2003.

[7] Necasky M., *"XSEM: a conceptual model for XML"*, 4th ACM International Asia-Pacific conference on Conceptual Modeling, Vol. 67, PP: 37 - 48, 2007.

[8] Lósio B. F., Salgado A. C., GalvÐo L. R., *"Conceptual modeling of XML schemas"*, 5th ACM International Workshop on Web Information and Data Management, PP: 102 – 105, 2003.

[9] Liu H., Lu Y., Yang Q., *"XML conceptual modeling with XUML"*, 28th International Conference on Software Engineering, PP: 973–976, 2006.

[10] Combi C., Oliboni B., *"Conceptual modeling of XML data"*, ACM Symposium on Applied Computing, PP: 467 – 473, 2006.

[11] Wu X., Ling T. W., Lee M. L., Dobbie G., *"Designing semistructured databases using ORA-SS model"*, 2nd International Conference on Web Information Systems Engineering, Vol. 1, PP: 171 – 180, 2001.

[12] Conrad R., Scheffner D., Freytag J. C., *"XML conceptual modeling using UML"*, 19th International Conference On Conceptual Modeling, PP: 558-574, 2000.

[13] Necasky, M., *"Conceptual Modeling for XML: A Survey"*, Tech. Report No. 2006-3, Dept. of Software Engineering, Faculty of Mathematics and Physics, Charles University, Prague, 2006.

[14] Choudhury S., Chaki N., Bhattacharya S., *"GDM: A New Graph Based Data Model Using Functional Abstraction"*, Journal of Computer Science and Technology, Vol. 21, Issue 3, PP: 430 – 438, 2006.

[15] Sarkar A., Choudhury S., Chaki N. and Bhattacharya S., *"Conceptual Level Design of Object Oriented Data Warehouse: Graph Semantic Based Model"*, International Journal of Computer Science (INFOCOMP), Vol. 8, Issue 4, PP: 60 – 70, 2009.

Adopting Knowledge Based Security System for Software Development Life Cycle

Jalal Alowibdi[1]
[1]Department of Computer Science, College of Engineering, University of Illinois at Chicago
jalowi2@uic.edu

Abstract— *The high-demand from the software industry led to the development of many Software Development Life Cycle (SDLC) models that help produce high quality software within budget and time constraints. Most of these SDLC models do not completely cover security as early as possible in the development cycle. Since security is a major concern to the users and the developers, adopting it at the early stages of the SDLC could help to ensure integrity, accessibility and confidentiality in future systems. It is still unclear how to achieve a perfectly secured software system by modifying the SDLC models. In this paper, the Knowledge Based Security System (KBSS) model is proposed to help in modeling and specifying security at all stages of SDLC in an effort to achieve a maximally secured software system. KBSS is a system that categorizes, clusters, monitors, alerts, and controls the Security Knowledge Management by the knowledge of the Security Expert Team, who are able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues.*

Keywords: computer security, software life-cycle, knowledge-based software engineering

1. Introduction

Software Systems (SS) play an integral role in many organizations and companies. SS often hold private and confidential information about organizations and companies who are using them. Ensuring the integrity, accessibility and confidentiality of such information in SS is a major concern in the software industry. Protecting the infrastructure of SS from threats and vulnerabilities could reduce the overall risk of "cyber attacks" [1]. Technically, including the security at the early stages of the Software Development Life Cycle (SDLC) helps reduce the overall risk of cyber-attacks as well as the cost and the time of restoring system operations that might arise after the deployment of the SS. Focusing on security at the early stages of the SDLC might produce well protected SS that prevent cyber-attacks on threats, vulnerabilities, and defects including man in the middle, denial of service, buffer overflow, SQL injections, and password attacks. It is critical and hard to guarantee secured SS all the time. However, applying a "Learn from Mistakes" approach could ensure just that. The approach is to build a Knowledge Base Security System (KBSS) that covers the most common threats, vulnerabilities and defects available up-to-date in the industry. Understanding these security issues, including abuse cases, as early as possible in the SDLC could help the developers to devote appropriate attention to security issues.

In this research, we seek to study different models that have been used in the industry. In addition, we attempt to model the security requirements using KBSS in order to provide up-to-date security functions and issues. These security functions and issues must consider all stages of SDLC as Iterative and Incremental Development Process (IIDP). This paper is organized as follows: Section 2 provides the related work in security at the early stages. Section 3 introduces the relation between security and the SDLC at all stages. The proposed model of KBSS within the SDLC at all stages, using IIDP model, discusses in Section 4. Section 5 discusses future work.

2. Related Work

A great deal of research has been conducted in the field of adopting security at the early stages of system development. Most of the research has been focused on security at the early stages of the SDLC, ideally at the requirements engineering stage. Very little research has been conducted on security at all stages. In this section, we seek to explore an initial concrete study of different models that have been used in the past. There are several studies related to security in the SDLC which can be summarized as follows:

McGraw [2] focuses on the security of the software at the early stages of the SDLC. Based on his research, he applies security concerns in the traditional waterfall model of the SDLC. Rigorously, it starts with the security requirements at the requirement stage to clarify the following questions: "How to protect? What to protect? and Whom to protect from?" Also, he describes the abuse cases. At the design and analysis stage, he identifies the security privilege and documents possible attacks, by a so-called risk analysis. At the code stage, he concentrates on implementation flaws, discovering the common vulnerabilities, and fixing bugs. Finally, he intends to determine feasibility of an attack by testing the overall functionality, called penetration testing.

Viega [3] used Comprehensive Lightweight Application Security Process (CLASP) to build security requirements.

Clearly, CLASP is "a set of process components that design to help the development teams to build and improve the security of the SS" [3]. The CLASP approach consists of four basic steps. These steps can be summarized as follows: (1) Identify system roles and resources; (2) Categorize the resources and the roles that make the process manageable (e.g., user-highly-confidential, user-confidential, user-low-confidential, system-private, and public); (3) Identify the resource interactions that relate to the roles and resource categorization; (4) Write the requirement specification. Also, the author defines the security services of CLASP as follow: Authorization, Authentication and Integrity, Confidentiality, Availability, and Accountability. His recommendation is to use the extension of the SMART+ requirements. Those requirements are Specific, Measurable, Attainable, Reasonable, Traceable, and + Appropriate.

Mead and Stenshney [4] defined the Security Quality Requirements Engineering (SQUARE) method, which "provides achievement for eliciting, categorizing, and prioritizing security requirements for the SS" [4]. Moreover, the SQUARE methodology builds security concepts into the early stages of the SDLC by documenting and analyzing security. The SQUARE method is divided into nine steps each having its own input, technique, participants and output. Rigorously, SQUARE starts with common security definitions and then identifies the goals of these security definitions. Once the organization has defined the common goals, it starts transforming them into deliverable requirements. It follows by choosing the elicitation techniques. These requirements can be categorized to meet the business goals. Next, risk assessment is performed with respect to the categorized requirements. The categorized requirements are prioritized. Then, this is followed by the inspection by the stakeholder as final step.

Romero-Mariona, Ziv, and Richardson [5] surveyed several approaches supporting the security requirements engineering in later stages of the SDLC. They explore six main areas to be considered in Later Stage Support (LSS) which represent support for the security requirements. First, security requirements integration provides support for integrating security requirements in the later stages of the SDLC. Second, constraint consideration expresses new constraints for security purposes. Third, security-oriented test cases provide validation of the security requirements. Fourth, test cases are developed alongside the security requirements. Fifth, the efforts to produce secure software are estimated. Sixth, the integration of other type of requirements focuses on non-security requirements.

Vetterling and Wimmel [6] combine a phase-oriented software development with Common Criteria (CC). They apply a so-called Target Of Evaluation (TOE) to assess the security requirements of the product based on the CC. The security requirement of the CC can be simplified into security functional requirements and security assurance requirements. Both are important to meet security objectives of the TOE. Moreover, the authors list the products of different assurance classes of the CC. Those classes are security target, configuration management plan, design and representation, life cycle documentation, test documentation, vulnerability assessment, guidance documents, and delivery and operation documentations.

Table 1 summarizes the previous survey by comparing the methodologies with respect to the following criteria: Defined Security Objectives (DSO), Adopted Method (AM), Target Development Process (TDP), Target Stage (TS), Iterative and Incremental Approach (IIA), Documented Security Functions and Issues (DSFI), Adopted Monitoring and Alerting Security System (AMASS), Used Heuristic Security Approach (UHSA), and Used Deterministic Security Approach (UDSA). Respectively, the proposed Knowledge Based Security System KBSS methodology is corresponding to all criteria.

Table 1: Summary compares the related works.

	M [2]	V [3]	M, S [4]	R, Z, R [5]	V, W [6]
DSO	YES	YES	YES	YES	YES
AM	NONE	SMART+	SQUARE	LSS	CC
TDP	waterfall	NONE	NONE	NONE	NONE
TS	ALL	REQ	REQ	REQ	REQ
IIA	NONE	NONE	NONE	NONE	NONE
DSFI	NONE	NONE	NONE	NONE	NONE
AMASS	NONE	NONE	NONE	NONE	NONE
UHSA	YES	NONE	NONE	NONE	NONE
UDSA	NONE	NONE	NONE	NONE	NONE

3. Security and the SDLC

Traditionally, many users think that a SS product is just code. we view a SS as a complete process methodology. It starts from the requirements stage and ends with deployment. The most famous SDLC is the waterfall model. There are many other models that have been used in the industry such as extreme programming, rapid application development, iterative and incremental development, the rational unified process, the spiral model, scrum development, and agile software development. However, most SDLC models do not completely cover security as much as they could. Additionally, the requirements stage is the earliest stage of the SDLC in all models. There are many sub-topics under the requirements to be considered within our proposed KBSS model as inputs and/or outputs of the requirements stage. Among them are stakeholder identification and system prototypes. These sub-topics might help in describing the security of the SS for our KBSS model. Good requirements engineering leads to a successful project within budget and time constraints. Security concerns are usually included in the non-functional requirements of a SS. However, our research focuses on security as part of functional requirements and constraints as well.

The increase of the SS vulnerabilities has made the software industry rethink security in their SDLC models. A major challenge in SS security is anticipating all plausible future attacks and ensuring that these attacks will not succeed. When an attack is discovered or expected, an appropriate solution is provided. The technical details of preventing the attacks are in continuous evolution. We have to extend our knowledge by understanding existent cyber-threats and by guessing future possible cyber-attacks. As long as there are malicious users trying to attack different SS using different techniques, there will be a challenge of providing 100% secured SS. In addition, we must also provide an iterative procedure to ensure SS security by incrementally expanding our KBSS.

3.1 Security Objectives and Attacks

Ideally, there are many security objectives to be considered, simulated and identified to have a secured SS. Each SS has its own security objectives that might be shared or not. Identifying the common objectives at the early stages of the SDLC lead to ensure the SS security. Getting familiar with the objectives will help to understand them from different aspects of our proposed KBSS. The common shared objectives are:

- Authentication is the concept of confirming the correctness of the identity of a user, and assuring the process of the system is trusted [7], [10]. An example of authentication is as follows: A user X plans to use the software system Y, Y must ask X for information that is provided by X and then verified by Y to authenticate X claiming identity is valid and trusted.
- Authorization is the concept of defining and specifying access rights or policies such as permit or deny to a user and/or a process of system [3], [7]. Ideally, authorization is a kind of approval or permission. For example, if the user X shows proper identification to the software system Y, Y authenticates X. Then, Y would grant authorization to X to access a set of certain information. Other information that is not related to X would not be allowed to access.
- Confidentiality is the concept of ensuring that the data is protected, secured, viewable and accessible only to the authorized user and preventing unauthorized user [7], [12]. An example of confidentiality is as follows: When the user X intends to access data D from the software system Y, Y must ensure that X has a permission to access D.
- Availability is the concept of ensuring the SS is reliable, runnable, serviceable and accessible to legitimate users [7]. Also, it is the concept of protecting the SS from unauthorized users who intend to make the SS unavailable. For example, when the user X decides to access the software system Y, Y must be available and ready for X.
- Accountability is the concept of ensuring that the interaction and communication between the users and the SS is trusted, traceable, reliable and accountable [7], [8]. For example, when the user X plans to use the software system Y, X would provide Y with information and Y would respond to X. The interaction and communication between X and Y must be accountable.
- Integrity is the concept of ensuring the quality, correctness and consistency of the data during the processing operations such as data transfer, data storage, data retrieval, and data quality. Also, it is the concept of protecting the data of SS from unauthorized users who intend to destroy the data during operation [7], [11]. For example, a user X intends to exchange data and information from the software system Y, the provided data must be accurate, correct, and complete. Y assures that the data does not get corrupted or deleted accidentally by unauthorized users who intend to change or delete the data during operation.
- Non-Repudiation is the concept of ensuring the communications of send and receive between the SS and legitimate user without being deniable [7], [9]. For example, a user X asks the software system Y for data D. Y sends X the required D, and X receives D from Y, where Y non-repudiation objective is to prevent X from denying receiving D in future.
- Non-Intrusion is the concept of preventing unauthorized user who attempts to break into the SS by identifying possible security breaches [13]. For example, a user X intends to compromise the software system Y using common known threats and attacks techniques that break Y. Y must assure that all the common fragility attacks techniques are perfectly secured and fixed toward preventing X's attempt, such as Buffer Overflows, SQL Injections, Design Flaws.
- Assurance is the concept of covering all security objectives by ensuring their workability and correctness. The SS is not going to have an error, failure or shutdown during an unauthorized user who attempts on attacking the SS by using common known threats and attacks. Assurance might raise a flag and alert the SS when the attack is detected [7]. For example, when the user X attempts to access the software system Y, Y must check its security using assurance A. A provides a checkpoint to check the correctness, stability, and workability of each security objective in Y. Then, A provides Y with information about the status of Y's security. Based on the provided information, Y decides whether to enable or disable X.

There are many relationships that can be defined between users, the SS, and security objectives in our KBSS. To apply security objectives correctly and produce secured SS, we must first have our SS to check its security objectives' correctness via assurance while the user is requesting to

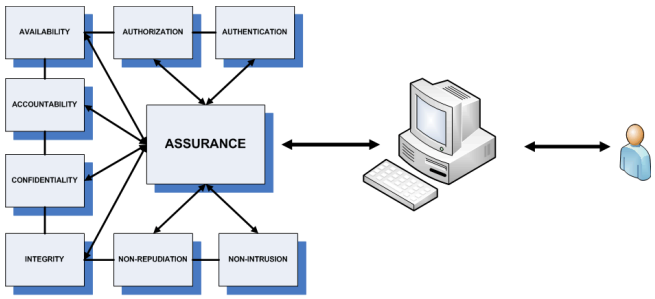


Fig. 1: Diagram illustrates the relationship between the user, the software system, and the security objectives, when the user intends to access a secured software system

access the SS. This methodology is preventing the user to access the SS until the SS approves its security consistency and correctness, ensuring loyalty of the user and preventing unauthorized user from accessing the SS. To simplify the scenario, when the user X requests to access the software system Y. Then, Y must check its security using Assurance A, which is the most significant security objective, that is responsible for checking Y's security correctness, stability and workability. Then, A checks X's identity using authentication, grants authorization to X to access certain information and makes data confidential and integral between X and Y. Otherwise, A rejects X. In the meantime, A ensures that the communications between X and Y is accountable, non-repudiative, and non-intrusive. Then, A provides Y with information about the status of Y's security. Finally, Y processes the provided information from A and decides whether to enable/disable X. When Y enables X to access the SS, Y keeps monitoring A iteratively while X is accessing Y. If A finds that one of the security objectives is compromised, A raises a flag and alerts Y. Figure 1 shows the relation between the user, the SS, and the security objectives.

We have introduced the most common objectives to ensure secured SS on one hand. On the other hand, we seek to clarify the possible attacks that compromise the SS security by taking advantage of the vulnerabilities of the SS. Technically, the attackers can target one or more security objective to compromise the stability of the SS. Knowing the common attacks and applying the appropriate security prevents the SS from being attacked. The assurance objective is responsible for assuring all security objectives of the SS where attacking one of them is correspond to the assurance. There are many examples of attacks against the weakness of the SS. Among them are: fraud attacks, phishing attacks, man in the middle, denial of service, worms, virus, spy-ware attacks, spam-ware attacks, cookies attacks, replay attacks, session attacks, hijack attacks, eavesdrop attacks, identity attacks and malicious attacks. The common possible and shared attacks are:

- Attack on authentication and authorization is the concept of attempting to break into the SS using the weakness of the authentication and authorization functions that mislead the identity and become an unauthorized user [3], [7], [10].
- Attack on confidentiality and integrity is the concept of attempting to access, read, modify and delete certain data [7], [11], [12].
- Attack on availability and accountability is the concept of attempting to make the SS or particular part of it unavailable to its users that leads the SS to be unaccountable. The accountability might result from every attack to the SS security objectives [7], [8].
- Attack on non-Repudiation and non-Intrusion is the concept of attempting to access the users logs by modifying and deleting the tracking data that misleads the trust between X and Y [7], [9], [13]. Also, we might consider the attack that takes advantage of newly discovered attacks on other organizations and uses it against other SS.
- Attack on assurance is the concept of attempting to compromise different objectives in one attack [7].

4. Knowledge Based Security System

The Knowledge Based (KB) of security is a special kind of database for Security Knowledge Management (SKM). SKM provides the most of our knowledge and up-to-date of having secured SS. SKM is the concept of systemically collecting, organizing, retrieving, enabling, storing and identifying the repository of security knowledge in an organization that can be available to other organizations.

Our proposed KBSS is a system that categorizes, clusters, monitors, alerts and controls the SKM by Security Expert Team (SET), who are able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues. KBSS uses heuristic and deterministic based approach that helps in providing the SS with numerous security functions and issues that can be beneficial to any organization. This system collects its data by SET using various resources such as security organizations' knowledge, security expert's knowledge and heuristic and deterministic security knowledge within the organization. KBSS keeps monitoring the security updates that are released by SET. Once there is a new released security, the KBSS starts releasing its alert. Precisely, KBSS uses different inputs and/or outputs behavior description. For example, in the requirements engineering stage, KBSS uses various forms for representing security behavior such as natural language, user stories, or requirements specifications. In the analysis and design stage, KBSS uses Unified Modeling Language (UML) for representing the behavior description of security and the same is applied to other stages. These inputs and/or outputs can be inserted into the KBSS by SET, who are able to find the security and provide the appropriate solution in different stages of KBSS. Also, KBSS is the bridge between

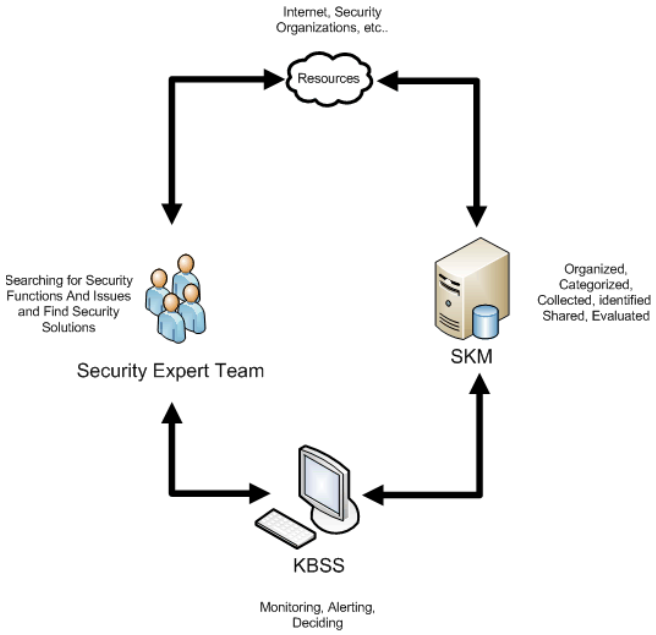


Fig. 2: Diagram illustrates the interactions between the SET and the SKM that controls the KBSS

the SET and the SKB, which holds all security knowledge to various domain. Figure 2 depicts the KBSS interactions.

Technically, there are four main actors. Each one of them depends heavily on his neighbor. The SET is an assigned team in the organization. Its responsibility is to collect, define, monitor, decide, and provide up-to-date security functions and issues with their appropriate solutions. The SET uses their knowledge of expertise and accesses the resources to collect security functions and issues, and insert them into the SKB via KBSS. Also, it is responsible for finding the appropriate solutions to security functions and issues that are not being recognized by the security organizations. Additionally, the resources can be classified as Internet's resources, security organizations' resources, Books' and Publications' resources, heuristic security within the organizations' resources and any valid security resources. SKM is special kind of database that systemically collects, organizes, retrieves, enables, stores and identifies the repository of security knowledge in an organization to be available to other organizations. SKM can be automatically accessing the resources and do the same job as the SET but with limited solutions. For example, the SET has the ability to find an appropriate solution that has not been recognized before where the SKM has not. We have to program the SKM to be an expert and intelligent to do decision as the SET. Keep in mind that the SET cannot access the SKM directly, which needs KBSS. KBSS is the main actor whose responsibility is to monitor, decide, and alert the development organizations that adopting it in the SDLC. Also, it is responsible for categorizing and clustering the SKM and provides recommendations to the organization with specific domain.

4.1 KBSS with SDLC

KBSS is trying to secure all well-known and expected security functions and issues within SDLC. Those functions and issues might be used as iterative and incremental approach toward SDLC. They can be simplified once they get discovered. Also, they must be alerted and adopted to the stage that the developer team has reached or just alerted and postponed to the next iteration. We adopt the KBSS in SDLC ideally with an Iterative and Incremental Development Process (IIDP). We are taking advantage of IIDP for adding security functions and issues with their appropriate solutions.

Simply, you can consider our KBSS methodology as a regular SS that keeps monitoring security functions and issues with appropriate solution. Once we start developing a SS, we must first add the domain of the SS to the KBSS. The KBSS is going to keep tracking our development processes with target concentration to the security perspective. When the KBSS is initialized with a specific domain, the KBSS is going to provide the developer with the most current initial security functions and issues. They might be represented as a natural language and provided at the beginning of the development processes. The KBSS will give appropriate security feedback highlighting issues and required functionality incrementally throughout the life cycle. As they are started at the requirements stage and end up with the evaluation stage, the KBSS moves forward iteratively and incrementally using IIDP compared to the waterfall model that stops at the last stage. The methodology adopts a checkpoint at every stage of the SDLC. Figure 3 shows the adaptation of the KBSS in SDLC.

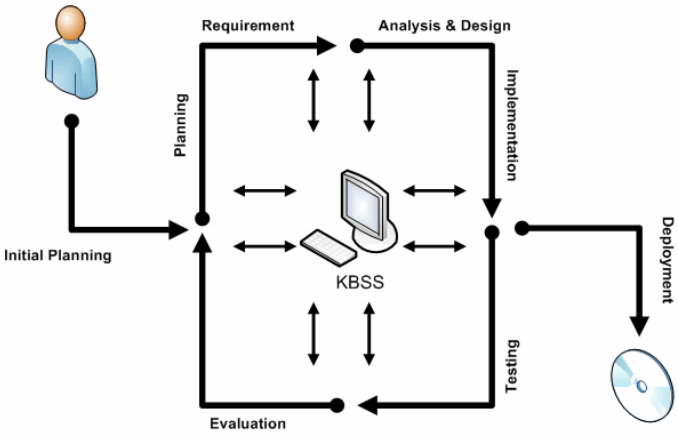


Fig. 3: Diagram illustrates the application of the KBSS to the Iterative and Incremental Development Process

Security updates are issued almost daily in response to new threats, which exploit previously-unknown vulnerabilities to attack SS. The newly-discovered vulnerabilities must be corrected in order to ensure that SS are completely secured. Many such updates could be issued while developing a given SS, requiring that the SS be modified appropriately in order for the SS to be secure. If a SS is developed with KBSS, a so-called security guard (checkpoint) will be evaluated at the end of each iteration of the SDLC. KBSS is a partially automated toolset that continuously monitors the security level of the SS under development after the development team specifies the domain of the SS. Simply, once we plan to develop a SS, the KBSS works as follows:

1) In the initial planning, we must add the domain of the SS to be developed to the KBSS.
2) The KBSS evaluates the domain, puts its first security checkpoint after the initial planning and extracts all the prospective security functions and issues with their appropriate solutions that might relate to the domain of SS. Also, it defines all the possible well-known and expected attacks and presents them to the developer in natural language for consideration. Passively, the KBSS automatically updates at discrete time intervals. whenever it gets updated with new security functions and issues, it starts alerting the developer of those functions and issues. The developer will consider them as an incremental and iteratively extension to the SS. Also, this checkpoint contains some of the issues that will be addressed in future stages of development. In addition, we could use one of the available tools, if there is any tool available, for converting the planning story to requirement specifications such as natural language to be used as requirements for this stage in our KBSS.
3) At time of processing the analysis and design stage, there is another security checkpoint for KBSS to evaluate the requirements security and provide the most recent security functions and issues with their respective resolutions. Also, it defines all the possible well-known and expected attacks that can be adopted to this stage such as abuses cases. For example, for accessing online banking system, the user must enter a valid user name and password. Once we have another user using the system from different machine, we must detect that and start using another type of verification such as security questions that previously set up by the user. This scenario is an example of security function and issue with its respective appropriate solution. Mainly, KBSS is using UML to represent security functions and issues at this stage. KBSS is an automated system that keeps updating him continuously. When KBSS detects an update, it directly raises a flag, alerts and contacts the developer team with that respective update. The developer will consider it as incremental and iterative feature to add to the SS. We might adopt one of the available tools [14], [15] for modeling the requirements in our KBSS. As a result, when there are new security functions and issues arise, KBSS will provide them to the developer team as UML to be added directly as incremental feature to the analysis and design stage.
4) At time of processing the implementation stage, there is another security checkpoint for KBSS to evaluate the analysis and design security and provide the most current security functions and issues with their respective appropriate solutions. Also, It defines all the possible well-known and expected attacks that can be adopted at this stage such as securing all objects and classes of the codes we implemented. KBSS is using algorithms or pseudo code to represent security functions and issues at this stage. Also, we could use one of the available tools [16], [17] that converting UML to code in our KBSS to validate the security releases issued at this stage.
5) At time of processing the testing stage, there is another security checkpoint for the KBSS to evaluate the implementation security and provide the most current security functions and issues with respected resolutions. It defines all the possible well-known and expected attacked that can be adopted such as buffer overflows, SQL injections, and design flaws. The KBSS will consider the black box testing, white box testing and gray box testing as testing security solutions to the developer team. The black box approach occurs when we have no advanced knowledge of the infrastructure to be tested. However, the white box approach happens when we have advanced knowledge of the infrastructure to be tested. The gray box approach is a hybrid between the two. Also, we could have the KBSS to provide the developer team with some tips, advises and techniques on testing and evaluating security at the testing stage.
6) Then, at time of processing the deployment stage, there is another security checkpoint for the KBSS to evaluate the testing security and provide the most current security functions and issues with their respective appropriate solutions. It defines all the possible well-known and expected attacks to the SS functionality that can be adopted at this stage. Also, KBSS is going to check if all the previous security functions and issues with their respective resolutions have been correctly applied. Otherwise, KBSS is going to flag, alert, and contact the developer team with the missing security functions and issues with their respective resolutions in order to be applied before deploying the SS.
7) Then, the first incremental released of the SS is deploying with respect to all aspects of security con-

cerns. Also, the KBSS automatically keeps tracking the updates with discrete time interval. When there are updates, it starts alerting the developer with those updates to be considered as incremental development processes.

8) Finally, for any release for security functions and issues to respective domain, we need to do step 2 to 6 repeatedly on one hand. On the other hand, we are going to adopt some of the available tools that are going to convert the planning stories security to requirements specification such as a natural language. Then, they are going to be modeled to UML and then to implementation codes. Also, they are going to provide the developers with tips, advises, and techniques on testing based on the provided UML. Finally, they are going to be provided to the developer team as incremental releases that are ready to use.

4.2 KBSS Objectives

The KBSS is successfully achieved to produce a secured SS when adopts its methodology at all stages of the SDLC ideally, as IIDP. Also, the KBSS is categorizes, clusters, monitors, alerts and controls the SKM, that contains many security functions and issues with their appropriate solutions to different domains, using the knowledge of the SET. In addition, the KBSS keeps tracking the security concerns within the SDLC using IIDP model. The KBSS can be incrementally added the security functions and issues with their appropriate solutions to the SS development as being developed and iteratively rework on the security concerns. KBSS is automatically monitoring and alerting the developers with security functions and issues as soon as they are released and discovered. Assuming the developers of the SS have valid subscription to our KBSS and their domains are registered in KBSS, KBSS is security solutions that can be beneficial to organization and any other organizations, who are planning to provide secured SS.

5. Conclusion

We have identified many major security objectives. To have secured SS, the SS must adopt security objectives correctly. We have realized that the attackers mostly target the weakness of security objectives. KBSS can discover and resolve the weakness of the security objectives. Moreover, we have recommended KBSS that categorizes, clusters, monitors, alerts and controls the SKM by the SET, who is able to identify, collect, organize, manage, retrieve, provide and store all aspects of security functions and issues. Additionally, we have offered evidential model of adopting the KBSS with SDLC ideally with IIDP. The basic idea of the adaptation is to have security checkpoints. Those security checkpoints help feeding SS with the most current security functions and issues with appropriate solutions.

In future work, we plan to use Bayesian Probabilistic System (BPS) for organizing KBSS. BPS helps ensuring the possibilities of what part of the system will be most likely to be attacked. Ideally, adding most of the knowledge of our common cyber-attacks against cyber-protects in BPS will help to provide connections between domains, the common cyber-attacks, and cyber-protects. The BPS will find the risk of possible attacks to specific domain.

References

[1] J. Wang, H. Wang, M. Gup, and M. Xia, "Security Metrics for Software Systems," in *47th Annual Southeast Regional Conference (ACM-SE 47)*, Article 47, ACM 2009.

[2] G. McGraw, "Building Security in Software Security," in *IEEE Security and Privacy Journal*, vol. 2, no. 2, pp. 80–83, IEEE 2004.

[3] J. Viega, "Building Security Requirements with CLASP," in *Workshop on Software Engineering for Secure Systems-Building Trustworthy Application*, ACM 2005.

[4] N. Mead and T. Stehney, "Security Quality Requirements Engineering (SQUARE) Methodology," in *Workshop on Software Engineering for Secure Systems-Building Trustworthy Application*, ACM 2005.

[5] J. Romero-Mariona, H. Ziv, and D. Richardson, "Later Stages support for Security Requirements," in *Richard Tapia Celebration of Diversity in Computing Conference: Intellect, Initiative, Insight, and Innovations*, pp. 103–107, ACM 2009.

[6] M. Vetterling and G. Wimmel, "Secure Systems Development Based on the Common Criteria: The PalME Project," in *Special Interest Group on Software Engineering*, pp. 129–138, ACM 2002.

[7] L. Ma and J. Tsai, "Attacks and Countermeasures in Software System Security," in *Handbook of Software Engineering and Knowledge Engineering*, Volume III, 2005.

[8] A. Yumerefendi and J. Chase, "Trust but Verify: Accountability for Network Services," in *11th Workshop on ACM SIGOPS European Workshop*, Article 37, ACM, 2004.

[9] M. Hwang, L. Li, and C. Lee, "A Key Authentication Scheme with Non-Repudiation," in *SIGOPS Operating Systems Review*, vol. 38, no. 3, pp. 75–78, ACM 2004.

[10] D. Mellado, J. Rodriguez, E. Fernandez-Mendine, and M. Piattini, "Automated Support for Security Requirements Engineering in Software Product Line Domain Engineering," in *International Conference on Availability, Reliability and Security*, pp. 16–19, IEEE 2009.

[11] L. Corman, "Data Integrity and Security of the Corporate Data Base: The Dilemma of End User Computing," in *SIGMIS Database*, vol. 19, no. 3–4, pp. 1-5, ACM 1988.

[12] P. Bennison and P. Lasher, "Data Security Issues Relating to End of Life Equipment," in *International Symposium on Electronics and the Environment Conference Record*, pp. 317–320, IEEE 2004.

[13] Z. Li, A. Das, J. Zhou, "Theoretical Basis for Intrusion Detection," in *Sixth Annual IEEE SMC on Information Assurance Workshop*, pp. 184–192, IEEE 2005.

[14] K. Subramanian, D. Liu, B. Far, A. Eberlein, "UCDA: Use Case Driven Development Assistant Tool for Class Model Generation," in *16th International Conference on Software Engineering and Knowledge Engineering*, pp. 324–329, Banff, Alberta, Canada 2004.

[15] R. Gaizauskas, H.M. Harmain, "CM-Builder: An Automated NL-Based CASE Tool," in *15th International Conference on Automated Software Engineering*, pp. 45–53, IEEE 2000.

[16] F. Do Nascimento, M. Oliveira, M. Wehrmeister, C. Pereira, F. Wagner, "MDA-based Approach for Embedded Software Generation from a UML/MOF Repository," in *19th Annual Symposium on Integrated Circuits and Systems Design*, pp.143–148, ACM 2006.

[17] W. Harrison, C. Barton, M. Raghavachari, "Mapping UML Designs to Java," in *15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, pp. 178–187, ACM 2000.

Construction of Quality Prediction Model based on Peer Review Performance Data

M. Komuro[1] **and N. Komoda**[2]
[1]Engineering Support Center, Hitachi Solutions, Ltd., Tokyo, Japan
[2]Graduate School of Information Science and Technology,Osaka University, Osaka, Japan

Abstract— *Firstly, a model is constructed which describes the relationship between the number of defects detected by tests and those detected by peer reviews. Secondly, a method is given to transform this model into a prediction model. Finally, the quality prediction model obtained by this method is validated through actual project data. The resulting model can predict the number of defects at testing phase with* $\pm 18\% \sim \pm 24\%$ *relative error.*

Keywords: Process Improvement, Peer Review, Process Performance, Quantitative Model, Quality Prediction

1. Introduction

It is important to know the quantitative effect of process improvement activities in order to correctly and effectively implement the improvements. Furthermore, if we can predict the effect of the improvement from the performance data of currently on-going processes, we can avoid the possible future trouble or more effectively perform the processes. For example, in CMMI(Capability Maturity Model Integration)[1] which is a standard of process improvement model this kind of mechanism is called process performance model and is considered as an important characteristic of high maturity[1].

Peer review is known to be a simple, effective, and inexpensive way of detecting and removing defects from software work products. Peer review is conducted not only on source code but also on various documents. It is known that peer review has both the quality improvement effect and cost reduction effect [2][3].

When a project has some quality problem, often it is not realized until the testing phase. However, it is too late or very costly to improve the quality in the testing phase. If we have some mechanism to predict the quality at the testing phase from the peer review performance data at the upper stream phases, it would be useful to prevent this kind of failures.

In this paper we propose a method to construct a model to predict the number of defects at the testing phase from the peer review performance data. We validate the resulting model through real world project data.

2. Effect Evaluation Model

We assume that the Waterfall lifecycle model is employed and sequential lifecyle phases are used for software development projects. Let us denote these phases as p_j $(1 \leq j \leq n)$ and further assume that the last several phases are allocated for testing. Since we are primarily interested in defect removal by peer reviews and we will not use each of these testing phases separately, we treat these several testing phases as one single phase denoted by a phase symbol T. We also assume that the phase just before the phase T is allocated for programming and denote it as a phase symbol P as illustrated in Fig. 1.

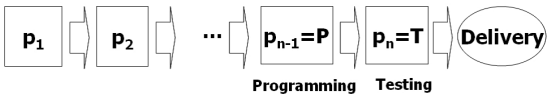


Fig. 1: Software development phases

Let $f(p_j)$ be the number of defects detected at the j-th phase p_j. When $p_j = T$, $f(T)$ is the number of defects of the program detected by testing. When the phase p_j is equal or earlier than P, $f(p_j)$ is the number of defects detected by peer reviews. The workproduct under review is source code in $p_j = P$ and development documents such as specification and design documents in the earlier phases.

Let us define $d(p_j)$ by $d(p_j) = f(p_j)/\Sigma_{k \geq j} f(p_k)$ and call it as the defect removal rate at phase p_j. Intuitively $d(p_j)$ represents how intensively the peer review is conducted at phase p_j, comparing with the performance of later phases.

If we conduct peer reviews intensively at some phase p_j so as to increase the defect removal rate, then the quality of the products later than p_j would be improved. Therefore it is natural to expect the defect density at testing phase will decrease. The following 'Effect Evaluation Model' represents this expectation in a quantitative way.

(Effect Evaluation Model)
For any phase p_j before the testing we have

$$\log(test.defect) = a_1 \log(size) + a_2 d(p_j) + \beta_1 + \cdots + \beta_n + c, \quad (1)$$

where $test.defect$ is the number of defects detected at the testing phase, $size$ is the size (KLOC) of the developed program, $d(p_j)$ is the defect removal ratio at phase p_j, a_1, a_2, c are constants, and $\beta_1, \cdots, \beta_n$ are terms representing the effect of factors other than peer review performance. It should be noted that defects are mainly removed through

[1]CMMI is registered in the US Patent and Trademark Office by Carnegie Mellon University.

peer reviews at phase p_j which is before the testing and that $d(p_j)$ represents the quality enhancement effect of these peer review activities.

This model assumes linear relationship between $\log(test.defect)$ and $\log(size)$ like COCOMO quality estimation model[4]. In our previous paper [2] it is shown that $d(p_j)$ affects exponentially on the defect density at testing phase. Although this Effect Evaluation Model deals with $test.defect$ directly rather than the defect density at the testing phase, we still assume that $d(p_j)$ affects exponentially on $test.defect$. Hence, $d(p_j)$ appears directly on the right hand side without $\log$ transformation. This can be viewed as a representation of a practical observation [5][6] that performance of peer reviews in upper stream phases strongly affects the quality in lower stream phases.

We validate this model by regression analysis using actual project data which are collected from a business division in a software development company. We refer to this division as 'the target organization' in the rest of this paper. These projects in the target organization build Web based software systems using Java language. We will use 35 data out of these projects excluding those which have missing records on some of development phases because they order suppliers to build a part of the development system.

The target organization has been improving their peer review processes [3][7]. A standard set of checklist items for peer review is established and used by projects through tailoring. This selection of checklist items defines what kind of defects should be detected at peer reviews.

We apply Generalized Linear model[8][9] to this validation. In our case, $test.defect$ is not a continuous but a discrete random variable derived from counts of defects. For continuous variables, the distribution of the residuals is often the normal distribution, for which classical linear model taking the least square sum of residuals is suitable. The distribution of the defect counts is different from the normal distribution and often the Poisson distribution is assumed[6][10].

Generalized linear models use maximum likelihood based on the selected distribution for the residuals to measure the goodness of the fit instead of the least square sum of the residuals. It is also possible to specify link functions to transform the response variables in Generalized Linear models and the canonical link function is determined for each residual distribution. For example, the canonical link function for Poisson distribution is the logarithm $\log(x)$ [9].

Since the response variable is already transformed by the logarithm in the Effect Evaluation Model, we take the logarithm for the link function.

Two kinds of variables are used for the β term.

(a) Risk evaluation results
These are dummy variables with values in the set $\{0, 1\}$ obtained from the evaluation results of risk analysis. In the target organization the risks are evaluated and given a numerical value between 1 and 5 as defined in standard risk evaluation sheet. We select about 30 check items expected to be related to the product quality. If the evaluation is 4 or 5, the risk is judged to be high and specific action is taken to monitor or mitigate the risk. We introduce a dummy variable for each item representing whether the evaluation is 4 or 5.

(b) $\log(base.size+1)$: logarithm of base system size
Often the software system to develop has some base system, i.e., the previously developed system on which the current development is based. Let us denote the KLOC of the base system as $base.size$. Qualitatively this variable represents the amount of experience on the developing system. As with the variable $size$ of development size, we take the logarithm of this variable. We add 1 to $base.size$ to prevent it from becoming negative infinity.

We first select the variables which have 5% statistical significance and then add or delete the variable checking whether the addition of the variable decreases residual deviance significantly. Since the addition of a dummy variable is logically equivalent to the introduction of a case classification, we carefully avoid adding too many dummy variables compared to the number of data points.

In the target organization the following symbols are used to represent development phases p_j's. BS: a phase to develop basic specification, FS: a phase to develop functional specification, DS: a phase to develop detailed Design, P: a phase for programming. These development phases are deployed in this order right before the testing phase. We conduct the linear regression analysis for each phase and get the result shown in Table 1. The line with the phase symbol $BS + FS + DS$ shows the result of regression analysis considering the three phases BS, DS, and FS as one development phase in which document reviews are conducted. We use quasi-Poisson distribution instead of Poisson distribution, because we observed over dispersion. That is, the residual deviance is much greater than the one expected from the degrees of freedom.

Table 1: Validation result

p_j	β **term**	**p-value**	**coeff.**	**pseudo** R^2	**#**
BS	LB	0.046	-2.43	96.7%	10
FS	CO,DE	6.6×10^{-8}	-2.13	96.6%	34
DS	LB, AC	0.037	-0.83	91.8%	30
BS+ FS+DS	DE	0.038	-0.92	89.8%	35
P	RC, DD	3.4×10^{-5}	-1.86	92.6%	34

The explanatory variables selected for β terms are the following. LB: logarithm of KLOC of base system, i.e., $\log(base.size+1)$, CO, DE, AC, RC, and DD are dummy variables representing the risk about customer organization, development environment, determination of system

architecture, requirements documentation, and delivery date respectively. We limit the number of dummy variables no greater than 2 for each regression equation.

The column under '#' shows the number of data points i.e., projects used for each regression. The reasons why the number of projects differs for each phase is because some projects skip some phases. Especially in the target organization there are many projects which skip the earliest phase BS and start from FS. There are two cases for this as follows. (1) Projects developing the system with a base system may not need to execute BS again. (2) The customer or another company performs the upper stream phases and a project in the target organization develops the system based on the requirements made in these upper stream phases.

The p-value shown in Table1 is that of $d(p_j)$ and the pseudo R^2 is McFadden's pseudo R^2. It shows the percentage of determination by log likelihood instead of the percentage of determination by sums of squares in the usual linear regression model. In every phase the value of pseudo R^2 is about 90%, which can be an evidence that the Effect Evaluation Model is valid. On the other hand, the occurrence of overdispersion suggests that there are some missing explanatory variables. However, we do not add dummy variables any more, because the number of data points are limited to at most 35 as shown above.

We also conduct simple linear regression analysis for each explanatory variable with a response variable $\log(test.defect)$. As a result, it is found that $\log(size)$ has the largest pseudo R^2 value. The result of this regression is as follows. p-value: smaller than 2×10^{-16}, pseudo R^2: 83.0%, regression coeffient: 0.92, standard error of regression coefficient: 0.06.

This result shows that the prime factor to determine the behaviour of $\log(test.defect)$ is $\log(size)$. It follows that we can get a approximated value of $test.defect$ by a formula of the form $test.defect = A\, size.^{0.92}$ Or more simply a formula of the form $test.defect = A'\, size$ can be used, judging from the value of the standard error.

3. Construction and Validation of Quality Prediction Model

In this secion we will construct a model to predict the number of defects in testing phase based on the Effect Evaluation Model described in the previous section.

3.1 Iterative Prediction Model

Our idea to construct prediction model is to consider the Effect Evaluation Model as an equation for $test.defect$ and solve it by the iteration method. Since the Effect Evaluation Model is established through regression analysis, it contains an error term. However, in order to explain our idea clearly, we ignore the error term for a while.

For the sake of simplicity, let us first consider when $p_j = P$. Then the defect removal rate at the phase P is given by $d(P) = f(P)/(f(P)+test.defect)$. Let us consider the right hand side as a function of $test.defect$ and denote it as $d_P(test.defect)$. That is, $d_P(x) = f(P)/(f(P)+x)$. Note that $d_P(test.defect) = d(P)$ holds. Then the following proposition holds.

(Construction of Iterative Prediction Model)
Assume that the equation of the Effect Evaluation Model

$$test.defect = size^{a_1} exp(c + a_2 d(P) + \beta_1 + \cdots + \beta_n) \quad (2)$$

holds without an error term and that the absolute value of the coefficient a_2 is smaller than 4.

If td_0 is a good enough approximating value for $test.defect$, then

$$size^{a_1} exp(c + a_2 d_P(td_0) + \beta_1 + \cdots + \beta_n) \quad (3)$$

will give a better approximation of $test.defect$ than td_0.
[Proof] Let ε be the difference between td_0 and $test.defect$, i.e., $td_0 = test.defect + \varepsilon$ and assume that ε is small enough.

Let us denote $size^{a_1} exp(c + a_2 d_P(x) + \beta_1 + \cdots + \beta_n)$ as $g(x)$. Then $g(test.defect) = test.defect$ holds by the assumption. It follows that

$g(td_0) - test.defect = g(test.defect + \varepsilon) - g(test.defect)$.
Expanding the first term in Taylor series, the last expression is equal to the following. $g'(test.defect)\varepsilon + (higherterms)$, where $g'(x)$ denotes the first derivative of the function $g(x)$. Ignoring the higher terms, we thus obtain

$|g(td_0) - test.defect| = |g'(test.defect)\varepsilon|$
By a straightforward calculation we get

$|g'(test.defect)| = |a_2| |\frac{f(P) test.defect}{(f(P)+test.defect)^2}|$.
However, by the well-known inequality of arithmetic and geometric means, we have

$\sqrt{test.defect\, f(P)} \leq \frac{test.defect + f(P)}{2}$.
Thus we obtain

$|g(td_0) - test.defect| \leq |\frac{\varepsilon a_2}{4}| < |\varepsilon|$
because of the assumption $|a_2| < 4$. This inequality shows that $g(td_0)$ is a better approximation of $test.defect$ than td_0. [QED]

As shown in the 'coeff.' column in Table 1 the condition $|a_2| < 4$ in the proposition holds for every development phase for the projects in the target organization. Therefore this construction can be applied to the data in the target organization.

The intuitive reason why $size^{a_1} exp(c + a_2 d_P(td_0) + \beta_1 + \cdots + \beta_n)$ gives a better approximation than td_0 is because the latter explicitly uses the information about the peer review performance. While the above proof is given under the asumption that the equation of the Effect Evaluation Model holds without an error term, we can expect $size^{a_1} exp(c + a_2 d_P(td_0) + \beta_1 + \cdots + \beta_n)$ is still a better approximation if td_0 does not use the informaion about peer review performance.

Under the asumption that the equation of the Effect Evaluation Model holds without an error term, we could repeat the

above procedure to compute the value of $test.defect$ more and more precisely. We call the prediction model obtained by this iteration as *Iteratvie Prediction Model*.

3.2 Quality Prediction Model

Now we know that the Iteratvie Prediction Model gives an approximating value of $test.defect$. However, the iteration of regression analysis could be very complicated and the resulting model would be hard to maintain. In this section we will show a way to 'transform' the Iterative Prediction Model into another model involving only one iteration. In order to do this, first we actually construct a model implementing the idea of 'iteration' and investigate its property.

1) Selection of initial solution
   We choose an initial solution to be proportional to the development size (KLOC), i.e., a solution of the form $test.defect = k\ size$, where k is the regression coefficient. As mentioned in the end of section 2, $log(test.defect)$ and $log(size)$ have a strong positive corelation with regression coefficient close to 1. This would be one of the natural selections.
2) Second approximation
   Using the above initial solution、we compute the right hand side of the Effect Evaluation Model.
   $a_1 \log(size) + a_2 d_P(k\,size) + \beta_1 + \cdots + \beta_n + c$
   $= a_1 \log(size) + a_2 \frac{f(P)}{f(P)+k\ size} + \beta_1 + \cdots + \beta_n + c$
   $= a_1 \log(size) + a_2 \frac{1}{1+k\,size/f(P)} + \beta_1 + \cdots + \beta_n + c.$

It should be noted that although the second term in the last formula is not a linear function but a rational function of $k\,size/f(P)$, it can be approximated well by a linear function within some bounded domain. In fact, as shown in Fig. 2 the function $y = 1/(1+x)$ decreases very slowly as x increases. It can be well approximated by a line if you limit the domain of x to an interval $(1/2, 2)$, for example.

The expression $k\,size/f(P)$ is obtained by approximating $test.defect/f(P)$. Since both $test.defect$ and $f(P)$ represent the number of defects found in the program, their values should not be too different. Therefore the value of $k\ size/f(P)$ is expected to be within some bounded domain and the line approximation mentioned above is meaningful. When we actually apply our model, we will choose data whose defect density is limited within certain interval. We will validate the resulting models by actual project data in 4.1 and 4.2. , where we limit $k\ size/f(P)$ to intervals $(1/2, 2)$ and $(1/4, 1)$ respectively.

From this argument we obtain the following formula.

$\log(test.defect)$
$= a_1 \log(size) + a_2' size/f(P) + \beta_1 + \cdots + \beta_n + c.$

It should be noted that the constant k appeared in the Iterative Prediction Model is included in the regression coefficient a_2' in this formula. It follows that we need to conduct regression analysis only once.

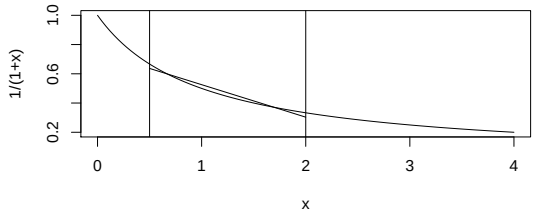


Fig. 2: Graph of y=1/(1+x) and a line approximation

The above argument for the phase P of programming can be applied similarly to the phase DS of detailed design which is performed just before P. More precisely an equation of the following form can be employed as the Quality Prediction Model:

$\log(test.defect)$
$= a_1 \log(size) + a_2 size/f(DS) + \beta_1 + \cdots + \beta_n + c.$

In this case the proposition (Construction of Iterative Prediction Model) in 3.1 will be modified as follows.

(Construction of Iterative Prediction Model (for DS))
Assume that the equation of the Effect Evaluation Model

$$test.defect = size^{a_1} exp(c + a_2 d(DS) + \beta_1 + \cdots + \beta_n) \quad (4)$$

holds without an error term, where the function $d_{DS}(x)$ is defined as $d_{DS}(x) = f(DS)/(f(DS)+x)$.
If td_0 is a good enough approximation for $test.defect$ whose error is within $\varepsilon > 0$, then

$$size^{a_1} exp(c + a_2 d_{DS}(d_0) + \beta_1 + \cdots + \beta_n) \quad (5)$$

will give an approximation of $test.defect$ such that the absolute value of its error is smaller than $|a_2\varepsilon|/4$.

The number of defects at the programming phase $f(P)$ is counted for the source code just like $test.defect$. Thus we can take an initial solution of $f(P) + test.defect$ to be proportional to the development size $size$ (KLOC).

The rest of the argument is the same as that for the programming phase P. As a result, we obtain

$\log(test.defect)$
$= a_1 \log(size) + a_2 size/f(DS) + \beta_1 + \cdots + \beta_n + c$

as a formula for the Quality Prediction Model.

Furthermore we can apply this argument to one combined phase BS+FS+DS. This phase is considered to be located right before the programinng phase, too. In these three phases BS,FS,and DS, peer reviews are conducted for documents. Hence, peer reviews in these phases are expected to have some common characteristics.

Let us summarize the resulting models for coding and document peer reviews.

(Quality Prediction Models)
The following equations hold.

1) In case of coding review,
   $\log(test.defect)$
   $= a_1 \log(size) + a_2 size/f(P) + \beta_1 + \cdots + \beta_n + c,$

2) In case of document review,
$\log(test.defect)$
$= a'_1 \log(size) + a'_2 size/(f(BS)+f(FS)+f(DS)) +$
$\beta'_1 + \cdots + \beta'_n + c'$,

where $a_1, a_2, c, a'_1, a'_2, c'$ are constants, $\beta_1, \cdots, \beta_n$, $\beta'_1, \cdots$, β'_n are terms representing factors other than peer review performance.

It should be noted that the right hand side of the Quality Prediction Models are actually computable at the P or DS phases. In fact at these phases the development size $size$ is either already determined or almost determined and precisely predictable.

Furthermore, in the target organization baselines for peer review perfomance (standards value and permissible ranges) are established and peer review processes are controlled with them[7]. The measures for these baselines include review rate and defect density. The Quality Prediction Models help a project to realize how many defects should be detected by peer reviews in order to achieve the project's quality objective (given in terms of defect density at the testing phase). The baselines and peer review process control with them are used to ensure enough defects are detected by peer reviews.

4. Validation by Actual Project Data

The Quality Prediction Model proposed in the previous section is derived under the hypothesis that the equation of the Effect Evaluation Model holds without the error term. In reality this hypothsis is not true. However, we can expect the Quality Prediction Model is valid as a regression equation. In order to show this, we conduct regression analysis on actual project data. We use the same 35 project data as used for the validation of the Effect Evaluation Model. The results of the regression analysis are given below.

4.1 Case of Coding Review

In the target organization projects define their quality objectives through 'bag rate' which is the defect density at the testing phase T. We select the condition to limit the line approximation described in 3.2 to be that the defect density at the programming phase P should be between $1/2$ and 2 times of the project's quality objective.

The rationale for this choice is that $test.defect/f(P)$ should take values around 1 as explained in 3.2. Actual width of the interval is determined so that we can keep enough precision while we can employ as many data as possible. Data from 23 projects satisfy this condition out of 35 projects.

As a result, we obtain the following regression equation and results.

$$test.defect = 6.08 size^{1.003} exp(-0.0403 \log(base.size+1) + 0.616 size/f(P) - 0.597 PM - 0.496 SF),$$

null deviance: 948.95, residual deviance: 52.196, degree of freedom: 17, pseudo R^2: 94.5%. Here, $base.size$ represents the size of the base system (KLOC). We add 1 to this value and take the logarithm as we did for the Effect Evaluation Model. The explanatory variables PM and SF are dummy variables concerning risks about project leader's experience and freedom of system developement respectively. As in the Effect Evaluation Model we use quasi-Poisson distribution instead of Poisson distribution, because we observed over dispersion.

In this equation $size/f(P)$ has a positive regression coefficient. It follows that if you increase $f(P)$ while keeping the value of $size$ then $test.defect$ will decrease, which means the product quality is enhanced. This property is practically important, because we can take remedy actions to increase the number of detected defects, if the predicted quality is not satisfactory. Furthermore it is possible to evaluate these activities quantitatively.

The term containing $base.size$ has negative coefficient, which suggests that if you have some experience of developing the same or related system, the quality will be enhanced. Both of the two dummy variables PM and SF have negative coefficients, which suggests that if the risk evaluation results are higher, then the quality will be enhanced. This result seems to be counterintuitive. However, as we explained in section 2, in the target organization projects are required to take some specific action, if the risk evaluation result is 4 or higher. There is a possibility that the quality is enhanced as a result of this action.

Fig. 3 is the scatter diagram of the prediction of the number of defects detected at the testing phase and the actuals. Most points are located near the diagonal line and the variation is not too large.

4.2 Case of Document Review

We construct a Quality Prediction Model for the combined phase $BS+FS+DS$. We select the condition to limit the line approximation to be as follows: The number of defects

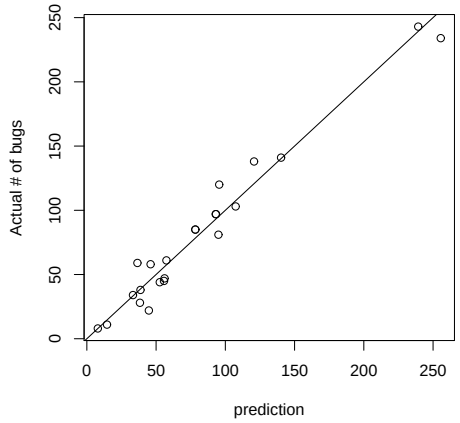


Fig. 3: Prediction of $test.defect$ from coding review data

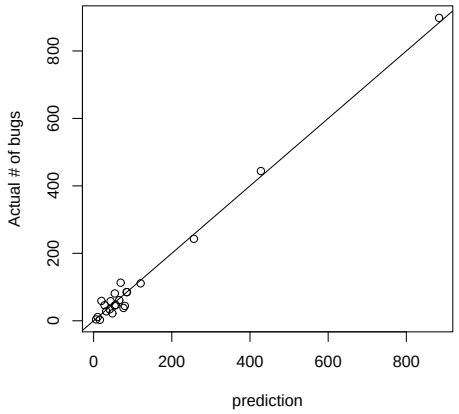


Fig. 4: Prediction of $test.defect$ from document review data

detected by peer reviews on documents should be between 1 and 4 times of the number of defects expected to be found at the testing phase. It should be noted that the latter number is calculated by the project's quality objective and estimated development size.

Since the numbers of defects over three phases are summed up, both the upper and lower limits of the condition is greater than those of coding review. These limits are determined so that we can keep enough precision while we can employ as many data as possible just like those of coding review. In terms of the interval to limit the domain of the function $y = 1/(1+x)$, this means we take an interval $(1/4, 1)$. It should be noted that we take the inverse value, because $x = test.defect/(f(BS)+f(FS)+f(DS))$. Data from 22 projects satisfy this condition out of 35 projects.

As a result, we obtain the following regression equation and results.

$$test.defect = 6.73 size^{1.09} exp(1.81 size/(f(BS)+f(FS)+f(DS)) - 0.81RC - 0.39RP),$$

null deviance: 4030.46, residual deviance: 184.66, degree of freedom: 17, pseudo R^2 : 95.4%.

The explanatory variables RC and RP are dummy variables concerning whether the requirements are clearly documented and whether the current system development replaces the existing system which is developed by other companies.

We use quasi-Poisson distribution instead of Poisson distribution, because we observed over dispersion. The regression coefficient suggests that the quality will be enhanced, if we increase the number of defects detected by peer reviews in this case, too. Similary, there is a tendency that the larger the risk evaluation result is, the higher the quality will be.

Fig. 4 is the scatter diagram of the prediction of the number of defects detected at the testing phase and the actuals. Judging from this graph, the data point at the upper right might affect strongly on the regression result. We investigated the Cook's distance. However, every data has 0.5 or lower distance. We can not find any evidence that the data affects strongly.

Fig. 5 shows an enlarged graph of the projects which detected 300 or less defects. Comparing this graph with Fig.3 which is drawn on the same scale, you can see this graph has larger variation. This may be because the programming phase P is still left when a project just completes the $BS+FS+DS$ phase.

In fact, we found the following fact by investigating the data. There are two circled groups of projects in Fig.5. The projects in the group located below the diagnal line detected relatively more defects in coding reviews. On the other hand, the projects in the group located above the diagnal line detected considerably fewer defects in coding review. This observation would imply that if you have a bad result by the Quality Prediction Model for document reviews, you still have an opportunity to improve the quality by conduting intensive peer reviews in the coming P phase.

Since we have overdispersions for both coding and document reviews, there is a possibility that the Poisson distribution is not appropriate. We conduct the conformance testing for the regression residuals. As a result, we get the following p-values: 0.26 for coding reviews and 0.38 for document reviews. Therefore, the null hypothesis that the disribution is Poisson is not rejected for both cases.

4.3 Discussion on Precision

We evaluate the precisions of the predictions by cross validation method. That is, select one subset of project data from the data set being analyzed, construct a model with the data set excluding the selected data, apply the model to the selected data to get prediction, and measure the difference between prediction and actual for the selected data. We apply this procedure to the 23 data of coding reviews and the 22 data of document reviews. As a result, we get the following average relative error: 18.6% for coding reviews and 24.0% for document reviews.

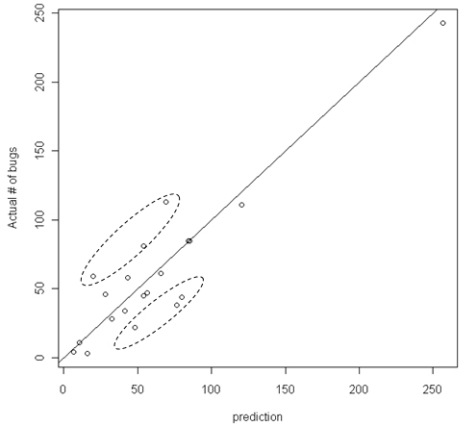


Fig. 5: Prediction from document review data (enlarged)

Since we can not find any publicized data for the precisions of estimates of the number of defects at testing phase, it is hard to evaluate our results objectively. However, by our experience, it is not unusual to have 10〜20% relative errors for an expert project manager to estimate the defect density at testing phase, even if he or she has experience of developing similar systems.

If one uses the Quality Prediction Model in this paper, one can get predictions with similar or a little bit worse precision even if one is not an expert or unfamiliar with the kind of system being developed. Thus, we consider our result is good enough to apply real-world development for the target organization and perhaps a similar quality prediction model could be derived, but with other weights and possibly a different set of input variables for other organizations.

5. Related Researches

There are several researches to predict quality from the project performance. Gaffney[11] and Kan [12] proposed to use Rayleigh curve to estimate the latent defects in the field. This method is validated by real world project data. However, the data they used are from very large size projects following rigid waterfall lifecycle phases which are not very popular these days. What they predict is not the number of defects at the testing phase but the number of latent defects in the field.

COCOMO II[4] is a family of various estimation models for software development including quality prediction. As for quality prediction, it employs log-transformed linear regression with several explanatory variables including development size. The explanatory variables other than development size use expert's opinion obtained by delphi method. Unlike the number of defects used in this paper, these variables do not use measured performance data.

Nakajima and Azuma[13] proposes a model to show how the quality cost will affect on overall development cost, based on 'Management model of total number of the defects.' A part of the formulation of their model is similar to ours at least formally. It appears that the estimating functions they propose are not validated through actual data. What they propose is a simulation model whose input are objective values of a project. A project compares the output of the model with actual project data and take corrective action if necessary. It does not predict the future outcome from the current project data.

Morgan and Knafl [14] proposes a regression model to predict residual fault density after the software system is released. Unlike the result of this paper, they do not predict the number of defects at the testing phase from the performace data at the upper stram phases.

6. Conclusion

We proposed a method to construct a model to predict the number of defects at the testing phase from the peer review performance data.

Firstly, we established a model to relate the number of defects at the testing phase with the defect removal rate of peer reviews at each development phase through regresiion analysis using Generalized Linear model. Secondly, we transform the model into another so that all the input data are available at an earlier development stage, i.e., peer review performance data and risk analysis results. Finally, we validated the proposed model with real world project data and found that the model can predict with $\pm 18\% \sim \pm 24\%$ relative errors, which we consider good enough for practical application.

This model has the following advantages: (1) It can predict the quality at early point in time. (2) It predicts the number of defects at the testing phase. And hence, we can know not only the quality improvement effect but also the cost reduction effect (through the reduction of rework). (3) Projects can take concrete corrective actions, if the prediction result is not satisfactory.

6.1 Acknowledgements

We would like to express our hearty thanks to Dr. Micheael Konrad of the Software Engineering Institute for his valuable comments.

References

[1] M. B. Chrissis, M. Konrad, and S. Shrum, *CMMI Guidelines for Process Integration and Product Improvement*. Addison-Wesley, 2003.

[2] M. Komuro and N. Komoda, "A model to explain quality improvement efect of peer reviews," *IEICE Trans. INF. & SYST.*, vol. E93-D, no. 1, pp. 43–49, 2010.

[3] M. Komuro, K. Otokozawa, and Y. Kimura, "Improvement of peer review process based on quantitative effect analysis of actual practices and performance of projects," *SEC Journal*, vol. 1, no. 4, pp. 6–15, 2005, (in Japanese).

[4] B. W. Boehm, C. Abts, W. Brown, S. Chlani, B. K. Clark, E. Horowitz, R. Madachy, D. J. Reifer, and B. Steece, *Software Cost Estimation with COCOMO II*. Prentice Hall, 2000.

[5] R. S. Kenett and E. R. Baker, *Software Process Quality*. Marcel Dekker, Inc., 1999.

[6] K. Ishikawa, *Introduction to Quality Management*. JUSE Press, Ltd., 1989, (in Japanese).

[7] M. Komuro, "Experiences of applying SPC techniques to software development processes," in *Proc. 28th International Conference on Software Engineering*, 2006, pp. 577–584.

[8] A. J. Dobson, *An Introduction to Generalized Linear Models*. Chapman & Hall/CRC, 2002.

[9] P. McCullagh and J. Nelder, *Generalized Linear Models*. Chapman and Hall/CRC, 1989, (second edition).

[10] W. A. Florac and A. D. Carleton, *Measuring the Software Process*. Addison-Wesley, 1999.

[11] J. E. Gaffney, Jr., "On predicting software related performance of large-scale systems," in *Int. CMG Conference*, 1984, pp. 20–23.

[12] S. H. Kan, *Metrics and Models in Software Quality Engineering*. Addison-Wesley, 2003.

[13] T. Nakajima and M. Azuma, "Models and a simulator for optimizing software quality process cost," *IEICE Trans. Inf. & Sys.*, vol. J91-D, no. 5, pp. 1216–1230, 2008, (in Japanese).

[14] J. Morgan and G. Knafl, "Residual fault density prediction using regression model," in *Proc. 7th International Symposium on Software Reliability Engineering*, 1996, pp. 87–92.

FWA – A Framework for Developing Web-Atlas Applications

Mark Rop and Yi Liu
Dept of Electrical Engineering and Computer Science
South Dakota State University
Brookings, USA
mkrop@jacks.sdstate.edu , yi.liu@sdstate.edu

Michael C. Wimberly
GISc Center of Excellence
South Dakota State University
Brookings, USA
michael.wimberly@sdstate.edu

***Abstract*—The increasing availability of geospatial datasets, software, and analysis tools has resulted in the expanded use of maps in the field of public health. Web-based mapping has the potential to greatly enhance the accessibility of these maps and other public health data to health practitioners and the general public. Framework of Web Atlas (FWA) is a novel framework for building web atlas applications that support organizing, managing, and providing access to large amount of map dataset to the public health. This paper presents the architectural design of the FWA, and gives a case study to show how to use it.**

Keywords-software framework; web atlas; software architecture

I. INTRODUCTION

The increasing availability of geospatial datasets, software, and analysis tools has resulted in the expanded use of maps in the field of public health. For example, data from earth-observing satellites is increasingly used to map the environmental risk factors associated with both chronic and infectious diseases [13], and to provide early warning of future disease outbreaks [3]. Widespread access to the Internet has fostered the development of novel techniques for disseminating and visualizing the resulting digital map products. These approaches range from simple images of static maps, to dynamic web-enabled GIS applications, to geo-browser applications such as Google Earth [4]. Potential use cases for these emerging technologies include visualization of large volumes of geographic data, interoperability among multiple applications, and modeling of environmental phenomena [5]. Web-based mapping has the potential to greatly enhance the accessibility of disease maps and other public health data to health practitioners and the general public.

Considerable attention has been focused on the creation of specific web mapping applications for public health [8], but there has been less emphasis on developing broader systems for organizing, managing, and providing access to large amounts of data through a variety of web mapping applications.

From the user side there is a need for a well-organized framework that allows users to access spatial information in a variety of forms. The organization of the geospatial datasets will depend of the geospatial data formats available.

By observing existing web-atlas applications and discussing requirements with the customers, we found that the following three issues challenge the producer's side for developing and maintaining the web-atlas applications:

i. The web contents have a dynamic nature. The maps might be updated frequently on a monthly basis, a weekly basis or even in a daily basis.

ii. Producers who need to make frequent changes to the web contents might not have web design or programming skills.

iii. The change to the organization structure of the maps might involve tremendous effort to rearrange all the web pages that disseminate the maps.

Thus, from the producer side there is a need for an easy-to-use application that will allow health organizations to update and manage large collections of electronic map products. This will involve adding new contents, editing existing contents, or deleting old contents. The producers or the web authors will be responsible for these operations.

To respond to the needs from both web-atlas users and producers, we have developed FWA – a novel framework that facilitates the development of the web atlas applications to be used to disseminate geospatial datasets and authoring of the web contents.

The paper focuses on the construction of FWA. Section 2 illustrates the architectural design of the FWA. Section 3 sketches the implementation of FWA and section 4 uses two case studies to illustrate how to use FWA in developing real world web-atlas. Section 5 concludes the results.

II. ARCHITECTURAL DESIGN

Based on the requirements for a couple of different web-atlas proposed by map producers, we make the following design decisions, which are true for all the FWA applications [7]. These design decisions will be the basis of the architectural design.

A. *Design decisions*

i. Each map is organized into a collection/category/record hierarchy. For example, collections could be defined as

types of diseases (e.g.,West Nile virus or malaria). Categories could include static maps (e.g., pdf and image formats), and dynamic web-enabled GIS applications (e.g., KML/KMZ formats). A map for West Nile disease in the pdf format will be presented in *West Nile* (collection)/*Static maps* (category).

ii. A map and its associated information can be added/deleted/modified.

iii. The producer's view and the end user's view are separated as figure 1 shows.

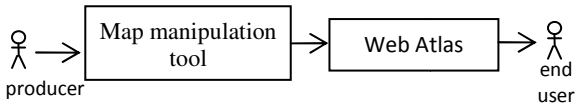


Figure 1. Separation of producer's view and end user's view

iv. The producer should be able to define and modify the categories and collections.

v. When the collection or category definitions are changed, the map records should be shipped to the updated collection/category upon the producer's request.

vi. The update to collections, categories, or maps should cause the changes to the web-based atlas.

B. Design of FWA arichtecture

A software framework is "a generic application that allows the creation of different applications from an application (sub)domain." [10] In object-oriented context, framework is "reusable design expressed as a set of abstract classes and the way their instances collaborate". [6] The difficulty part of framework design is the identification of the abstractions that can be tailored to specific applications within the framework domain.

In Schmid's [11] approach, within a certain framework domain, the variable and application specific aspects are abstracted as *hot spots* and the common aspects as *frozen spots.*

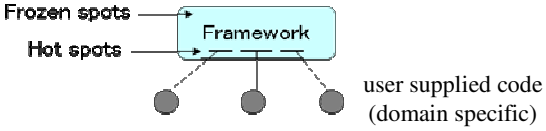


Figure 2. Hot spots and Frozen Spots

1) Frozen spots system

In the design of FWA, the common aspects are identified to construct the *frozen spots system.*

Based on the design decision (i), we use the hierarchy of collection/category/record to manipulate the maps. A collection can contain 0 to more categories. A category can contain 0 to more records, which holds the maps and their descriptive information.

Based on the design decision (iii), FWA is decomposed into *map manipulation* and *web atlas* for the first cut of the architecture. Further refinement is done on the map manipulation part by decomposing it into *Map storage*, *Map control* and *Web generation*. *Map Control* is designed for design decisions (ii) and (iv). It takes the user's inputs to add/modify collections, add/modify/delete categories, and add/modify/delete records. *Map Storage* stores the maps with the map hierarchy. *Map Control* feeds the data (maps) to *Map storage*. *Webpage generation* is designed for design decisions (v) and (vi). It retrieves the data from the *Map Storage* to make updated web pages for the web-based atlas. Figure 2 shows the top level FWA architecture with frozen spots added it.

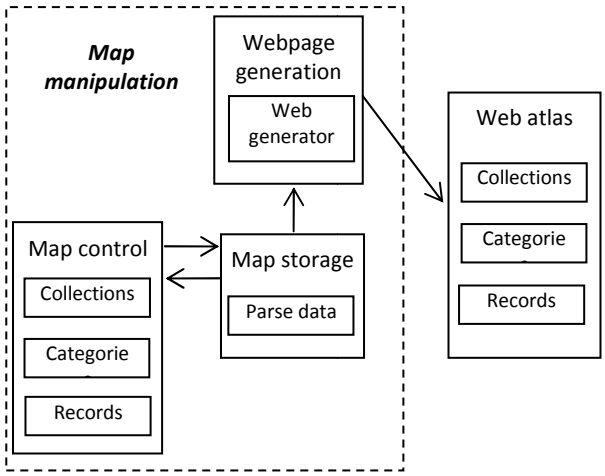


Figure 3. Top level FWA architecture with frozen spots

2) Hot spots system

Although the web atlas applications in the domain we focus on can share the modules shown in Figure 3, a web atlas application can also have unique features.

A web atlas application can have its own way of styling the web pages for hosting the web atlas, its individual location to store the maps in the *Map storage*, and its preference of the *Map control* user interface.

The variable aspects on webpage location and styling, map storage location, and map control user interface are addressed for constructing the *hot spots system.*

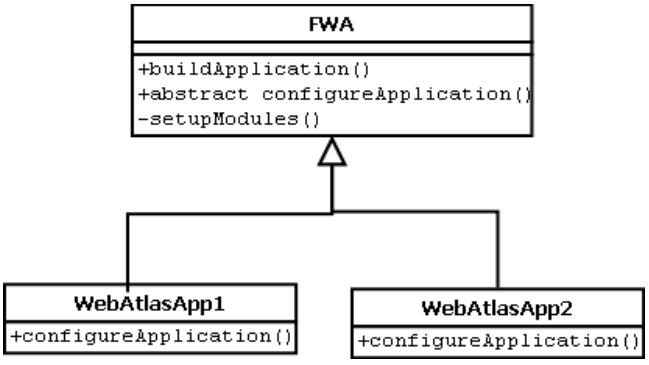


Figure 4. FWA's hot spots system

Template method pattern [2] is used in abstracting the three variable aspects. The *Abstract Template class* declares abstract primitive methods *configureApplication()*, which will be implemented by each concrete web atlas application to achieve its specificity. The template method *buildApplication()* invokes

the abstract method and set up the locations of *Map Controller*, *Map Storage*, and *Web Generator* and ship all the files with these three modules in frozen spots system to the concrete application.

3) FWA architecture

The FWA architecture (shown in Figure 5) is completed by connecting the frozen spots system and hot spots system.

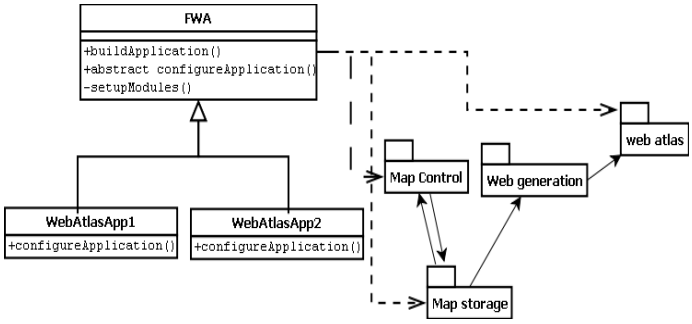


Figure 5. The architecture of FWA

III. FWA IMPLEMENTATION

The FWA has been developed mainly in PHP and AJAX. Three web atlases have been extended from FWA and currently are in use.

A. *Implementation of FWA frozen spots system*

1) Map storage

The maps are stored in the collection/category/record hierarchy. As mentioned in the Introduction section, one of the challenges for developing and maintaining web-atlas applications is that the change to the organization structure of the maps might result in rearranging all the web pages that disseminate the maps. For example, consider a situation where originally the producer makes all records held in collection directly. Later, there is a need to add a layer of categories (e.g., category1 and category2) to better organize the records. To respond to this challenge, we used XML due to its flexibility in presenting the information and it portability for transmitting the data. In the second phase, we plan to add search functions in FWA, and the usage of XML will also be beneficial for supporting the approaches of semantic search.

XML schema is used to describe the structure, define allowable document content and validate the correctness of data in the FWA XML document.

With the collection/category/record structure, a collection (e.g, *West Nile Virus*) contains 0 or more categories (e.g., *Historical West Nile Virus Data* and *Forecast 2011*), and a record contains a record title, a record description, a thumbnail of the record, and multiple maps (e.g., *Long-Term Hot Spots* record with 1 image map and 1 kmz file for download). Figure 6 shows a segment of the xml document.

```
<?xml version="1.0" encoding="UTF-8"?>
<EASTWeb
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="FWA.xsd" >
  …
  <collection id="West Nile Virus">
    <category id="Default">
    <category id="Historical WNV Data">
      <record>
        <title>Long-Term Hot Spots, South Dakota</title>
        <description>Local indicators of spatial association (LISA) …
        <thumbnail>thumb.png</thumbnail>
        <files filedescription=" Long-term hot spots, 2002-2007
              (.png)">lthp.png</files>
        <files filedescription="Long-term hot spots, 2002-2007
              (Google Maps)">lthp.kmz</files>
      </record>
    </category>
  </collection>
  …
</ EASTWeb>
```

Figure 6. Portion of the XML document for data storage

The *ReadData* class is developed to wrap the functions for accessing the XML map document. *get* and *set* methods are provided for accessing collection, category and record.

```
class ReadData
{
    /* class variables */
    private $_xml;
    public function ReadData($xml)
    {  $this->_xml=$xml;  }

    private function getCollections()
    {  $collection= Array();
       //Parse the xml and return all the collections
       return $collection;
    }
    private function getCategories($collection)
    {  $categories=Array();
       //Parse the XML and return the categories in the collection
       //Specified.
       return $categories;
    }
    private function getRecords($collection, $category)
    {
       $records= Array();
        if(isset($collection))
        {   //if collection is defined, get the records in the collections
            ……
        }
       else if(isset($category))
       {   //get records in the category specified
            …….
       }
       else
       {  //log the error
       }
        return $records;
    }
}
```

Figure 7. Accessing the map XML document

2) Map control

Map control is where a producer manipulates the maps through the collection/category/record hierarchy. Client side scripting and server side scripting are both involved. To

shorten the response time and avoid frequently reloading the web pages during interaction, AJAX is used to exchange the data between client and server asynchronously. Server side scripting is developed in PHP.

Collection, Category and Record have add, delete and edit functions. Interface *ItfControl* is introduced to declare these three functions to allow polymorphic access to Collection, Category and Record. Figure 8 shows the UML diagram of this design and Figure 9 shows the code of ItfControl and Figure 10 shows the partial code of how Collection implements the ItfControl interface.

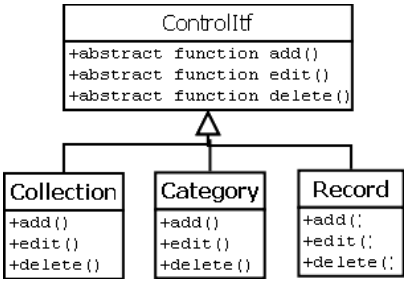


Figure 8. Collectio, Category, Record and the interface

```
abstract class ControlItf
{
  abstract public function add() { //define add operation}
  abstract public function edit(){//base edit function}
  abstract public function delete(){//base delete function }
}
```

Figure 9. .Map Control interface

```
class Collection extends ControlItf
{
   /* class variables */
   private $_collection;
   private $_xml;
   public function Collection($collection)
   { $this->collection=collection;          }
   public function add()
   {//override base class
    //implements the add functionality
   }
   public function edit()
   {//override base class edit
    //implements edit functionality
   }
   public function delete()
   {override base class delete
   //implements delete functionality
    }
}
```

Figure 10. Partial code of Collection class

3) Webpage generation

This module dynamically generates the web pages when a new collection or a new category is created. For example, the producer creates a collection named *West Nile Virus* and a category named *Historical West Nile Virus Data* under that, a web page for West Nile Virus and a web page for Historical West Nile Virus Data will be created on the end user's view and the new collection and new category will be added to the website navigation menu if it has one.

```
class WebGeneration
{
   public static function createCollection($collectionName)
   {
      //locate application collection template
      //Open and read file
      //dynamically write the destination collection file
   }
   public static function createCategory($categoryName)
   {
        //locate application specific category template
        //Open and read file
        //dynamically write the destination category file
   }
}
```

Figure 11. Web Generation

B. Implemenation of FWA hot spots system

The hot spots system contains an abstract class FWA (shown in Figure 12). This abstract class will be implemented in each concrete application.

```
abstract class FWA
{    protected $controllerPath;
     protected $storagePath;
     protected $generatorPath;
     protected $appname;

     public buildApplication()
     {          configApplication();
                // setup Map Controller
                setupModules($controllerPath, filePath::controller);
                // setup Map Storage
                setupModules($storgePath, filePath::storage);
                // setup Web Generator
                setupModules($generatorPath, filePath::generator);
      }

     abstract function configApplication();

     private function setupModules(&$appPath, $FWAPath)
     {       //path to unzipped FWA module
             $path = $FWAPath;
             $appname = str_replace(" ","",$appname);
             //path to destination application module
             $appPath="/".$appname."/".$path;
             if(!is_dir($appPath))
             {      mkdir($appPath);         }
             //bulding the module
             exec("cp $path $appPath");
     }
}
```

Figure 12. FWA abstract class

IV. USE OF FWA

It is very easy to use FWA to build a web atlas in the similar domain. All the developer needs to do is to (1) construct a website with the user's preferred layout and color

schema, and (2) implement the FWA abstract class. (1) could be done with any web editor, such as DreamWeaver. The following example presents the template code for implementing FWA for a concrete application.

EASTWeb [1] is a collaborative project involving scientists from South Dakota State University (SDSU) and the USGS Center for Earth Resources Observation and Science (EROS), along with partners from government agencies, and non-governmental organizations. Their research focuses on the application of geospatial technologies for mapping, risk analysis, and ecological forecasting of infectious diseases. The main purpose of EASTWeb is to provide public access to the various products that will be developed by the project, including geospatial datasets, maps, and forecasts.

The developer constructs a website for the end users to provide the information on background, updates on research activities, relevant web resources, and a spot for hosting the web atlas part. The website is shown in figure 13.

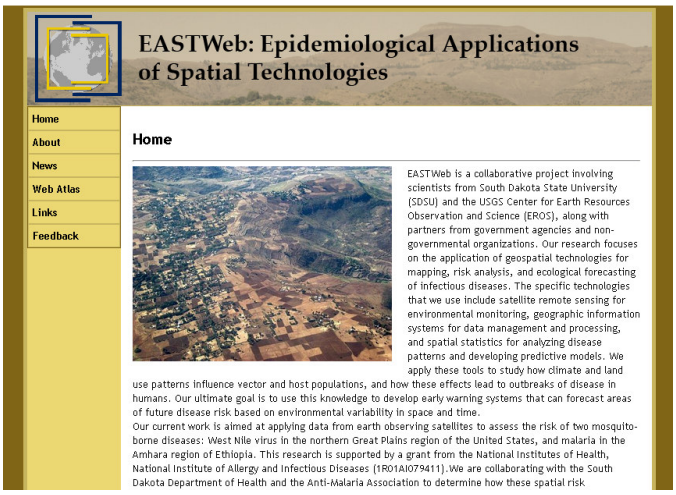


Figure 13. EASTWeb end user's view

The developer needs to allocate the locations (would be folders on the server) for holding the Map Controller and Data Storage. Once it is set, the developer implements the methods *configreApplication()* by given the application name to make the specific map manipulation tool for the producer and link the map data to the web atlas part of the website for the end users. Figure 14 shows the implementation. Figure 15 shows the producer's view of map manipulation tool and Figure 16 shows the end user's view of the web atlas.

```
class EASTWebApplication extends FWA
{
    public funtion EASTWebApplication();

    function configApplication()
    {       parent::appname= "EASTWeb";          }
}
```

Figure 14. Extending FWA for constructing EASTWeb

Figure 15. EASTWeb – producer's view

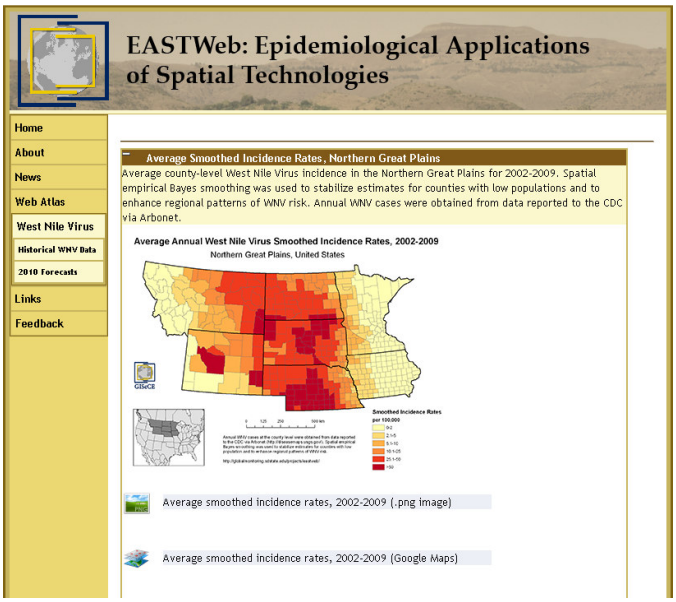


Figure 16. EASTWeb – end user's view

V. CONCLUSION

FWA is a framework that is designed and developed to support organizing, managing, and providing access to large amounts of geospatial data in the field of public health. The FWA can be easily customized to specific applications by being given category or collection definitions and the template of the web page layout.

FWA effectively facilitate the construction and maintenance of web-based atlas. Although the initial purpose of FWA is for the field of public health, the design of FWA architecture makes it flexible to be extended into building applications in other similar domains. We have already developed two more web atlases by using FWA in addition to the EASTWeb site described in section IV for forecasting

infectious disease. One of them is for providing geospatial information to support environmentally sustainable biomass production in the North Central Sun Grant Region [12]; and the other one [9] is for disseminating the geospatial data products resulted from the research on testing multiple working hypotheses about the environmental drivers of obesity.

In conclusion, we have found that the FWA greatly facilitates the dissemination of digital maps by allowing developers to quickly and easily implement a web atlas within a website, and by providing users with access to large amounts of geospatial information in a variety of formats. Given the experience of extending three web atlas applications from FWA, we believe FWA can be successfully and effectively used in constructing web atlases for multiple projects encompassing a variety of thematic areas.

ACKNOWLEDGMENT

This work is supported by NIH grant 1R01AI079411-01 "*An Integrated System for the Epidemiological Application of Earth Observation Technologies*".

REFERENCES

[1] EASTWeb project website.
http://globalmonitoring.sdstate.edu/projects/eastweb/. Last access date: 5/17/2011.

[2] Gamma, E., Helm, R., Johnson, R.,Vlissides J, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[3] Grover-Kopec, E., M. Kawano, R. W. Klaver, B. Blumenthal, P. Ceccato, and S. J. Connor. An online operational rainfall-monitoring resource for epidemic malaria early warning systems in Africa. Malaria Journal 4:6, 2005.

[4] Mills, J. W., and A. Curtis. Geospatial approaches for disease risk communication in marginalized communities. Progress in Community Health Partnerships 2: 61-72, 2008.

[5] Goodchild, M. F. The use cases of digital earth. International Journal of Digital Earth 1:31-42, 2008.

[6] Johnson, R. Frameworks Home Page.
http://st-www.cs.uiuc.edu/users/johnson/frameworks.html. Last accessed: 5/17/ 2011.

[7] Liu, Y., Rop, M., and Wimberly, M. C. On the construction of framework of web-atlas (FWA). Proceedings of ACM South East, 2010.

[8] MacEachren, A. M., Crawford, S., Akella, M., and G. Lengerich. Design and implementation of a model, web-based GIS-Enabled cancer atlas. The Cartographic Journal 45: 246-260, 2008.

[9] Obesity web site.
http://globalmonitoring.sdstate.edu/projects/obesity/. Last accessed date: 5/17/2011.

[10] Schmid, Hans Albrecht. Systematic framework design by generalization. Communications of the ACM. 1997.

[11] Schmid, H. A. Framework design by systematic generalization, In Fayad, M. E. , Schmidt, D. C., and Johnson, R. E., editors, Building Application Frameworks, pp. 353-378, Wiley, 1999.

[12] Sun Grant website:
http://globalmonitoring.sdstate.edu/projects/sungrant. Last accessed date: 5/17/2011.

[13] Wimberly, M. C., A. B. Baer, and M. J. Yabsley. Enhanced spatial models for predicting the geographic distributions of tick-borne pathogens. International Journal of Health Geographics 7:15, 2008.

Software Infrastructure for Grid Computing

O.O. Adesina and D.R. Aremu
Department of Computer Science, University of Ilorin, Ilorin, Kwara State, Nigeria.

Abstract - *Due to large data sets and accompanied large number of parameters being produced by high throughput techniques, it became necessary to develop high performance computers based on clustering technologies and high performance distributed platforms. Research efforts in achieving a high performance distributed platforms yield grid computing. However, grid is complex as a result of heterogeneous nature of the underlying softwares and hardware resources forming it. Managing these issues has become an important research and development challenges. In the literature, various implementation solutions have been proposed for managing heterogeneity in distributed systems. Programming language and platform dependency problems of these solutions are the motivation of this paper. We have proposed software infrastructure for managing the heterogeneity in grid computing environment. Our proposed solution is based on the standards and specifications of Web services framework.*

Keywords: Distributed systems, grid computing, heterogeneity, web service, and middleware.

1. Introduction

Grid computing [2][8][11][15] is becoming a de-facto technology for supporting the execution of large-scale, resource-intensive and distributed applications. It defines the combination of computers or clusters of computers across networks, like the internet, to form a distributed supercomputer. This computational infrastructure allows scientists to process complex and time consuming computations in parallel on demand. Grid infrastructures are based on a distributed computing model where easy access to large geographical computing and data management resources is provided to large multi disciplinary VOs (Virtual Organizations). However, appropriate software infrastructure for implementing the complex and heterogeneous nature of grid computing remains a big challenge. The aim of this paper is to present a software infrastructure model for implementing the complexities and heterogeneous nature of the grid computing environment. The presented implementation solution is based on standards and specifications of Web services framework [7].

The rest of this paper is organized as follows. Section 2 discussed the related work, while in section 3, we presented a generalized implementation strategy for the grid. In section 4, the Software Infrastructure model proposed for implementing the grid computing systems was discussed. Section 5 discussed the implementation plan for the presented model. Finally, section 6 presented the summary and future direction for this work.

2. Related Works

In the literature, various implementation solutions [1] have been developed to manage heterogeneity in distributed systems. Among these are CORBA, DCOM, Globe, and Java RMI. In the following subsections, we presented a detail review of each of the stated related implementation solutions.

2.1 Common Object Request Broker Architecture

CORBA [1][13] is an industry defined standard for distributed systems. An important goal of the Object Management Group, OMG with respect to CORBA was to define a distributed system that could overcome many of the interoperability problems, with integrating networked applications. CORBA's global architecture adheres to a reference model of the OMG. This reference model consists of four groups of architectural elements connected to the Object Request Broker (ORB). This ORB forms the core of any CORBA distributed system; it is responsible for enabling communication between objects and their clients while hiding issues of heterogeneity. Though no specific implementation has been tied to CORBA. However in its remote-object model, the implementation of an object resides in the address space of a server. Server objects and services in CORBA are specified in Interface Definition Language (IDL). This IDL provides a precise syntax for expressing methods and their parameters. CORBA interface is a collection of methods, and objects specify which interfaces they

implement. These interfaces are binary in nature and independent of programming languages.

CORBA implementation naturally accommodates extensions. Being an effort of committees, it has features and facilities in abundance. It is flexible in architectural models. CORBA is programming language, operating systems, and machine independent. It offers facility to find services that are available to a process. CORBA allows dynamic construction of invocation requests. Its flexibility in assigning interface identifiers, allows uniqueness in interface definitions within interface repository. It is a better platform for reusing legacy systems. CORBA is suitable for large web-enabled applications where performances under heavy client load are crucial. Although CORBA is programming language independent, however it is necessary to provide exact rules concerning the mapping of IDL specifications to existing programming languages. Till date, only few of these rules are available, because it is time and effort consuming. CORBA also illustrates that making a simple distributed system may be somewhat overwhelmingly difficult exercise.

2.2 Distributed Component Object Model

DCOM [1][14] originated from Component Object Model (COM). COM is the underlying technology of various Windows operating systems produced by Microsoft starting with Windows '95. Like all object-based systems, DCOM adopts remote-object model [1]. DCOM object is an implementation of an interface which can either be placed in the same process, as client on the same or remote machine. DCOM has only binary interfaces, such an interface is essentially a table with pointers to the implementations of the methods that are part of the interface. To define these interfaces, DCOM uses Microsoft IDL (MIDL), from which the standard layout for binary interfaces can be generated. These binary interfaces are programming language independent.

DCOM is a widely accepted middleware solution, with tens of millions of people using windows daily in networked environment. It is programming language independent, and requires no rules for mapping implementation to languages as CORBA does. DCOM also supports dynamic invocation of objects. It offers interface repository for storing and retrieving interfaces. To facilitate object activation, DCOM offers Service Control Manager (SCM) in conjunction with the Window registry. Due to the transient nature of DCOM's objects, garbage collection is less an issue. In spite of these benefits, DCOM has its problems. Among these is that it is not an effort of a committee. Based on this fact, it offers minimal set of core elements from which components and services are built. DCOM is an intricate system, because similar things can be done in different ways, and such that coexistence of different solutions is sometimes even impossible. It is platform dependent (i.e. Windows platforms). Passing object references, to another process in DCOM demands special measures, because its objects are transient by virtue of DCOM's object model.

2.3 Global Object-Based Environment

Globe [1][10] is an object-based system in which scalability plays a central role. All aspects that deal with constructing a large-scale wide-area system that can support huge numbers of users and objects drive the design of Globe. Fundamental to this method is the way objects are viewed. Like other object-based systems, objects in Globe are expected to encapsulate state and operations on that state. An important difference with other object-based systems is that objects are also expected to encapsulate the implementation of policies that prescribe the distribution of an objects state across multiple machines. Objects in Globe describe how, when, and where their state should be migrated and replicated. Unlike most other object-based distributed systems, Globe does not adopt remote-object model. Instead, the state of an object can be distributed and replicated across many processes.

Globe has great benefits and disadvantages. Its benefits are that, it can be used to support a huge number of users and objects spread across the internet, which is contrary to most other object-based distributed systems. Globe objects make decisions on how, when, and where its state should be migrated? They may also determine the security policies and implementation. Because the location service may return many contact addresses for an object, it does give options to select a contact address based on any selection criterion, such as distance or expected QoS. Objects contact addresses are flexible in specifications. This empowers clients to use any implementation, provided it obeys the rules guiding the protocol. However, the flexibility in contact address specifications comes with a price of having to make implementations for different local objects, and possibly for different operating systems and machine architectures.

2.4 Java Remote Method Invocation

Java RMI [1][5] adopts remote objects model as only form of distributed objects. Recall that a remote object is a distributed object whose state always resides on a single machine, but whose interfaces can be made available to remote processes. Interfaces are implemented in the usual way by means of proxy, which offers exactly the same interfaces as the remote objects.

Benefits of using Java RMI are enormous. The distinction between local and remote objects is hardly visible at the language level. It also hides most of the differences during a remote method invocation. Java RMI makes distribution apparent where a high degree of transparency is simply too inefficient, difficult, or impossible to realize. The complexity associated with marshaling proxy by converting its complete implementation into series of bytes, was addressed by generating implementation handle, specifying precisely the classes needed for constructing proxy. This makes Java RMI the most efficient of all object-based distributed systems. Though, Java RMI can hide most of the differences during a remote method invocation. However, primitive or objects involved in this process must be serializable; but platform-independent objects such as file descriptors and sockets can not be serialized.

3. Software Infrastructure Model for The Grid

Figure 1 below represents our proposed model for managing heterogeneity in grid computing environment. It is composed of group of clients and servers systems; service broker; Wide Area Network (WAN)/Local Area Network (LAN); and Application Programming Interfaces (e.g. inquiry and publisher APIs). Clients in the context of this model refer to group of applications or systems that accesses remote services on other computer systems, known as servers, with the aid of a network. In similar context, servers refer to a group of computer systems with computer programs or softwares running as services, and serve the need or request of other application programs (clients). It may or may not reside on the same machine (computer system) with the client. In the design of our model, we adopted different measures from the literature for managing heterogeneity. For instance, like all other object-based distributed systems, we adopted the remote-object model. Service implementation can either be placed in the same machine, or in a process at remote location as client application. Our model is centered on the implementation of interfaces. A service is an implementation of an interface; a single service can implement several interfaces simultaneously. In contrast to CORBA, DCOM and others that offer binary interfaces, our service interface was based on text document. These text documents are expressed in eXtensible Markup Language (XML) grammar. Interface definition is always convenient with a separate Interface Definition Language (IDL). In our model, Web Service Description Language (WSDL) [6] was adopted for interface definition. It is a standard format for describing all that is demanded by the client to bind to a service. The benefit of using textual interfaces is that interfaces are platform and programming language neutral. Once a WSDL of a service has been created, a service consumer must be able to find it, in order to be able to use it. This is known as discovery. Similar to interface repository in CORBA is the Universal Description, Discovery, and Integration (UDDI) [3]. UDDI project is an industry effort to define a searchable registry of services and their descriptions to facilitate automatic service discovery by consumers. Being an industry effort, we have adopted this as our service broker. Moreover, standardized packaging protocol for the messages shared by the applications is represented in Simple Object Access Protocol (SOAP) [4]. It is similar to CORBA. This specification defines a simple XML-based envelope for information being transferred, and a set of rules for translating application and platform-specific data types into XML representations. The design of SOAP makes it suitable for a wide variety of application messaging and integration pattern. The fundamental idea is that two applications, irrespective of operating system, programming language, or any other technical implementation detail, may openly share information using a simple message encoded in a way that both applications understand. Transport mechanism for enabling direct application-to-application communication on top of the network layer includes technologies like TCP, HTTP, SMTP, and Jabber [7]. Our choice of transport protocol is HTTP, because it is a standard transport protocol and provides ubiquitous firewall support. The network layer of our model is exactly the same as the network layer in the TCP/IP Network Model. It provides the critical basic communication, addressing, and routing capabilities for clients and servers in grid computing environment.

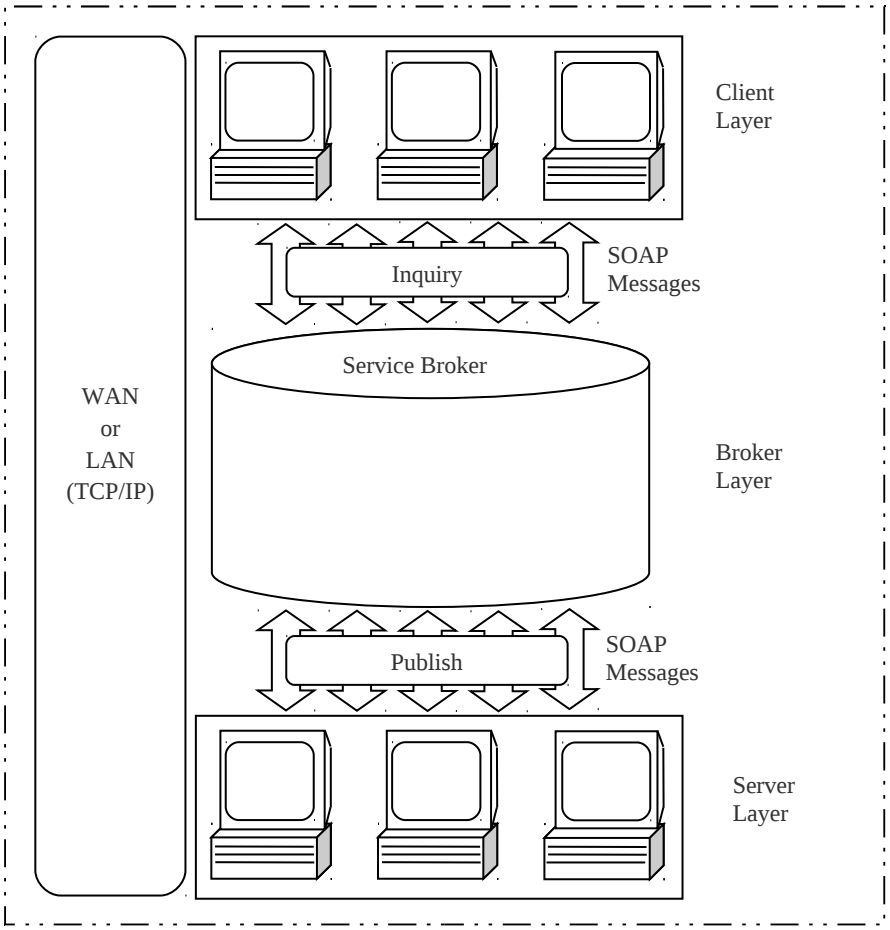


Figure 1: Grid Computing Model

4. Internal Structure of Grid Systems

We presented in Figure 2 below, the general organization of a grid system. At the client side, the software is kept to the minimum. The WSDL specifications of a service are simply compiled to a stub that serialize invocation requests into SOAP request messages, and deserialize the corresponding reply messages into results that can be handed back to the invoking client. Instead of generating a service-specific stub, a client can also dynamically invoke service through the Dynamic Invocation Interface (DII). Using stub or proxy is sometime referred to as static invocation; because the stub must know the remote interface at compile time and client must first obtain a reference to service implementation. The reference implementation for the stub is obtained by instantiating the service implementation class. Clients may also access a service using DII instead of static stubs. Unlike the static invocation, which requires that the client application include client stub, DII enables a client application to invoke a service whose data types are unknown at the time the client were compiled. This allows a client to discover interfaces at runtime, and invoke methods on objects that implement those interfaces. In the same vein, server side software, besides service implementation includes service listener, and eXtensible Markup Language (XML) [9] Processor. The service listener is the endpoint of the service, where the service can be invoked by consumer. XML processor is a software that handle validation/parsing of the incoming SOAP request, transformation of SOAP to domain-specific XML representation (i.e. removal of SOAP envelope), deserialization of the XML representation to in-memory tree that can be modified

by the service implementation; and vice-versa. Figure 2 accommodates implementations of client application and service implementation in different programming languages. Similarly, client and server operating systems may differ. Service broker in Figure 2 is service discovery software in the proposed grid computing environment. It provides publication and inquiry services for servers and clients respectively. Being a service provider in the proposed environment, its internal structure is the same as the server.

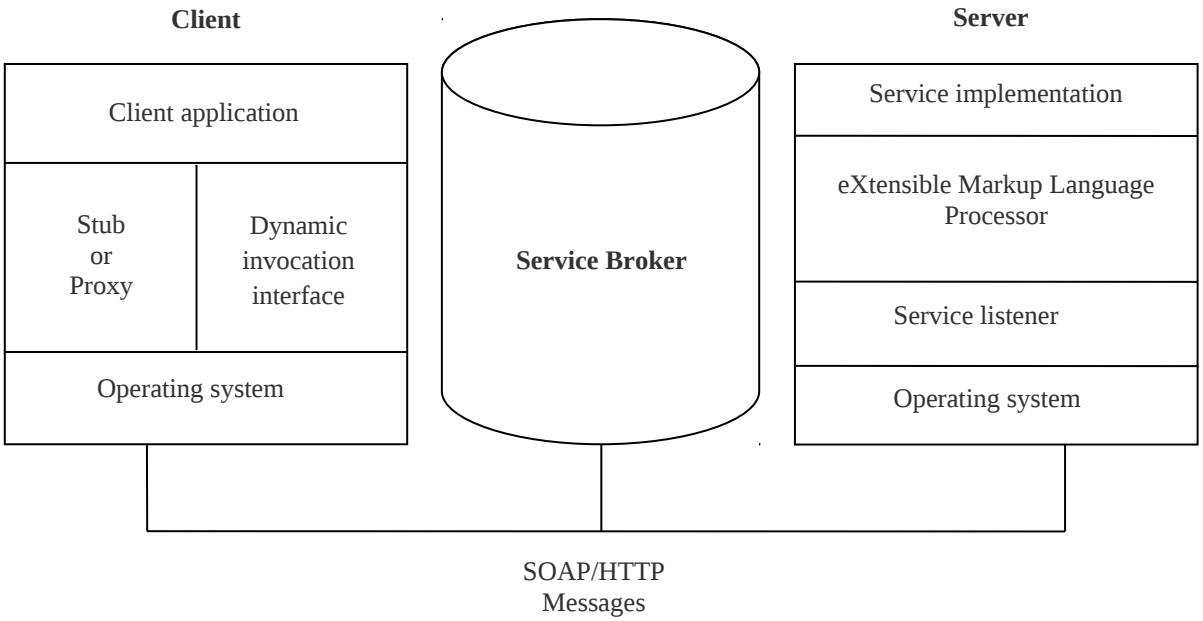


Figure 2: The General Organization of a Grid System

5. Implementation

The development and consumption of Web services involves the following phases: definition, implementation, deployment, description, and consumption [12]. The term service definition is used to refer to the abstraction that defines the publicly surfaced view of the service. It is represented as an interface that exposes the service's operations. The methods in the interface must have valid data types as arguments and return types. If the data types are user-defined, then appropriate serializers and deserializers must be provided; to facilitate marshaling and unmarshaling to and from the corresponding XML representations. Any request sent with incorrect information at runtime will generate a SOAP fault, because it will not be able to unmarshal the XML. Following this is the service implementation. This is the concrete representation of the abstract service definition, which is implemented using class in object oriented languages, and data structure in procedural languages. Once a service is defined, it is essential that it is deployed. Service deployment happens at runtime. A service endpoint is a junction where the SOAP message is received and the response dispatched. It is the physical entity exposed to service consumers that essentially services client requests. Once the service executes the client request, the endpoint is responsible for packaging and sending it back over HTTP. If a service has been defined, implemented, and deployed as endpoint, its description is important for consumption purposes. Based on the service definition, the WSDL document describes the service, its operation, arguments, return types, and the schema for the data types used in them. At this point, the service is ready for consumption by a service consumer. A service consumer represents the abstraction of the entity invoking the facilities of an existing service. Consumers can do this in two forms, as shown in the figure 2. These are proxy and DII as discussed in figure 2 above. The service broker in our model went through the stages of service development and consumption. It is a typical web application that is exposed as a Web service

through inquiry and publication interfaces. Publication interface allows service provider to register business information about the service(s) they offer. Once the service has been published, clients can enquire about the published services, to obtain binding information.

6. Conclusion

Grid computing has become a standardized technology for supporting the execution of large-scale, resource-intensive, and distributed applications. The benefit in adopting grid includes the ability to: pool heterogeneous computing resources across administrative domains and dispersed locations; run large-scale applications that outstrip the capacity of local resources; improve resource utilization; or collaborate applications [8]. Since grid is a distributed system, the probability is high that it is heterogeneous in nature. However, existing solution for managing the heterogeneity issues in distributed systems are inefficient for grid computing. In this paper, we have performed extensive evaluation of various solutions for managing heterogeneity issues in distributed systems. Moreover, we have presented a software solution that is independent of programming languages and platforms to be used for grid. The phases of implementing our proposed software infrastructure are presented in section 5. Realizing an operational service broker that is adaptive, efficient, secured, and fault tolerant is a big challenge.

7. References

[1] Andrew Tanenbaum & Marteen van Steen. "Distributed Systems Principles and Paradigms'. Prentice-Hall Inc., 2002.

[2] Belapurkar Abhijit., Chakrabarti Anirban, Ponnapalli Harigopal, Padmanabhuni Srinivas, Sundarrajan Srikanth & Varadarajan Niranjan. "Distributed Systems Security: Issues, Processes and Solutions". John Wiley & Sons, Ltd., 2009.

[3] Tom Bellwood, Luc Clement, David Ehnebuske, Andrew Hately, Maryann Hondo, Yin Leng Husband, Karsten Jaruszewski, Sam Lee, Barbara Mckey, Joel Munter & Claus von Reigen. "UDDI Version 3.0 Technical Report". Available at http://uddi.org/pubs/uddi-v3.00-published-20020719.htm, 2002.

[4] Box E., Ehnebuske D., Kakivaya G., Layman A., Mendelsohn N., Nielsen H., Thatte S. & Winer D. "Simple Object Access Protocol (SOAP) 1.1". Available at http://www.w3.org/TR/SOAP, 2002.

[5] Bryan T. "Java RMI: Remote Method Invocation", IDG Books Worldwide, Inc., 1998.

[6] Christensen E., Curbera F., Meredith G. & Weerawarana S. "Web Services Description Language (WSDL) 1.1", W3C Note, Available at http://www.w3.org/TR/WSDL, 2001.

[7] Dough Tidwell, James Snell & Pavel Kulchenko. "Programming Web services with SOAP". O'Rielly, 2001.

[8] Dubitzky Werner. "Data Mining Techniques in Grid Computing Environment". John Wiley and Sons, Ltd., 2008.

[9] Elliotte Rusty Harold. "XML 1.1 Bible", Wiley Publishing, Inc., 2004.

[10] Homburg P.,van Doorn L., van Steen M., Andrew S. & de Jonge W. "An Object Model for Flexible Distributed Systems". In Proceedings 1st Annual ASCI Conference, 69-78, May 1995.

[11] Kesselman Carl & Foster Ian. "The Grid: Blueprint for new Computing Infrastructure". Kaufmann, 1998.

[12] McGovern J., Tyagi, M., Stevens, M. & Matthew, S. "Java Web Services Architecture". Morgan Kaufmann Publishers, 2003.

[13] Object Management Group. "The Common Object Request Broker: Architecture and Specification, Revision 2.4.2". OMG Document formal/00-02-33, Object Management Group, 2001.

[14] Platt D. "The Essence of COM and ActiveX: A programmers Workbook", Prentice Hall, 1998.

[15] Stefano Michael. "Distributed Data Management for Grid Computing". John Wiley & Sons, Inc., 2005.

Intelligent Conceptual Message Routing in Enterprise Service Bus (ESB)

Amir Massoud Bidgoli[1], Payam Nabhani[2]

[1]Islamic Azad University Tehran North Branch. drbidgoli@gmail.com
[2]Islamic Azad University Science And Research Khozestan Branch. Ahwaz,P.Nabhani@gmail.com

Abstract A lot of researches have been done in Message Routing in Service Oriented Architecture (SOA). The main part of SOA is called Enterprise Service Bus (ESB) and services should be routed in this part. Message Routing is the primary functionality of ESB and message routing algorithms are very crucial. Many algorithms and method are announced till now but most of them are tight coupled to message using static routing and working in flat or liner structures. Intelligent Conceptual Message Routing (ICMR) is working based on ordered pairs of conceptual service metadata which is issued by service provider and they are stored in a tree. The request of service consumer is translate into ordered pairs and then searched by intelligent agent will be able to find appropriate service. In our work we concentrate on the efficient routing method that can be find appropriate services by defining the parameters by requester on the fly.

Keywords SOA, ESB, Message Routing, Middleware, Service, Intelligent Routing.

1. INTRODUCTION

Enterprise applications are increasingly being architected in a service-oriented architecture (SOA) style, in which modular components are composed to implement the business logic. The properties of such applications, such as the loose coupling among the modules, is promoted as a way for an agile business to quickly adapt its processes to an ever changing landscape of opportunities, priorities, partners, and competitors. The proliferation of Web services standards in this area reflects the industry interest and demand for distributed enterprise applications that communicate with software services provided by vendors, clients, and partners. [1]

The growing complexity of enterprise applications has favored the adoption of the architectural principles behind the Service-Oriented Architecture (SOA). The wide range of functionalities available through-out the enterprise network, often from legacy applications or systems, is encapsulated in services at the points where the functionalities are implemented. Once some functionalities (whatever its underlying technology, implementation or other details) are available as services, they can be reused in other enterprise applications. An SOA favors the construction of applications around independent building blocks (services), which can be arranged in different ways to create new applications. To enable maximum flexibility in the combination of services, these should be stateless, such that successive invocations of a service are independent from each other. At the core of such an architecture, there is an ESB (Enterprise Service Bus) enabling the operation of the applications by allowing the services to be invoked and to communicate with each other in a location-independent fashion. The ESB may also interconnect services that use different protocols and data formats by leveraging mediation functions. At the core of an ESB implementation there is at least one form of middleware that establishes a lingua franca used by services in the exchange of information, regardless of the actual location or invocation requirements of other services. This middleware may use a number of mechanisms, such as message queues, publish/subscribe messaging or Web Services (WS). WS provide well-defined standards that make them ideal for interoperability.[2]

Part of SOA is the infrastructure that allows you to use services in a productive system landscape. This is usually called the enterprise service bus (ESB). There are different opinions about the exact role and responsibilities of an ESB. Part of the reason for the different understandings of ESBs is that there are very different technical approaches to realizing an ESB. To run SOA in practice, you need a way of calling services. This infrastructure is the technical back-bone of the SOA landscape (sometimes also called the “backplane”). [3]

It is the responsibility of the ESB to enable consumers to call the services providers supply. This sounds simpler than it is. Depending on the technical and organizational approaches taken to implementing the ESB, this responsibility may involve (but is not limited to) the following tasks:

- Providing connectivity
- Data transformation
- (Intelligent) routing
- Dealing with security
- Dealing with reliability
- Service management
- Monitoring and logging

These tasks may need to be carried out for different hardware and software platforms, and even for different middleware and protocols. The ESB’s main role is to provide interoperability. Because it integrates different platforms and programming languages, a fundamental part of this role is data transformation. Another fundamental ESB task is routing. There must be some way of sending a service call from a consumer to a provider, and then sending an answer back

from the provider to the consumer Depending on the technology used, and the level of intelligence provided, this task may be trivial, or may require very complicated processing. [3]

The ESB should be equipped with routing mechanisms to facilitate not only topic-based routing but also, more sophisticated, content-based routing. Topic-based routing assumes that messages can be grouped into fixed, topical classes, so that subscribers can explicate interest in a topic and as a consequence receive messages associated with that topic. Content based routing on the other hand allows subscriptions on constraints of actual properties (attributes) of business events. The content of the message thus determines their routing to different endpoints in the ESB infrastructure. [4]

Our work is about to define a new mechanism to routing the message through ESB. The mentioned method is a Conceptual Routing Message that could find the appropriate service intelligently.

The rest of this paper is organized as follow. Section 2 introduces related work of message routing .The proposed Conceptual Message Routing method is discussed in Section 3. Section 4 gives a sample built by using ICMR. Conclusion is made in Section 5.

2. Related Work

A. Content-Based Routing [5] [6] [7]

Content-based routing (CBR) is a message driven dynamic routing mechanism and considered as a necessary feature on ESB. In CBR, the routing path is determined by analyzing the message and applying a set of predictions to its content and it usually depends on a number of criteria, such as the message type, specific field value, existence of fields, and so on.

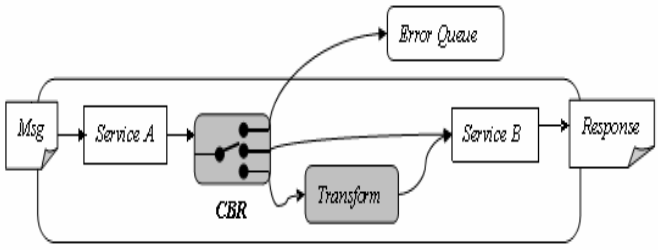


Fig. 1 CONTENT-based Routing on ESB

B. Pattern-Based Dynamic Routing [5] [8]

In conventional routing models, a service is usually maintained by single service container or unknown external server and their composition capacity usually depend on Web Services or BPEL, which are based on XML technology that has well interoperability but well-known performance drawbacks in high volume or intricate applications. Pattern-based dynamic routing (PBDR) [5] focuses on this issue and introduces application pattern (AP) concept to improve current solution. AP is the key to support the multi-service container responsible for service orchestration in PBDR since it defines a set of interoperable services and the relationship among them, so it can helps to obtain a routing path for message delivery. Generally, an AP instance maintains a steady composition application for PBDR module. Then by analyzing a received message, PBDR will choose an appropriate one and invoke the services involved in to process the message.

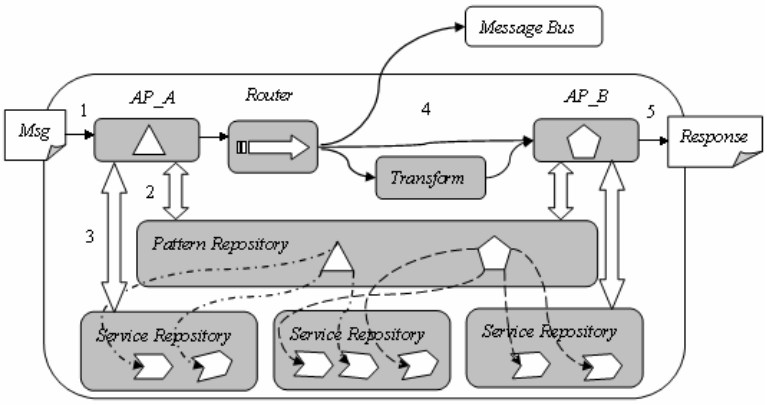


Fig. 2 Pattern-based Routing on ESB

Compared to CBR, PBDR (shown in Fig.2.) replaces the service node with AP and it has a pattern repository and service repository for AP and service registration. PBDR process a message in five steps:

1. When receiving a message, PBDR validate the message header and deliver it to an appropriate AP container (here is AP_A).

2. AP_A will select an AP instance form Pattern Repository by applying a matching predication to the message. The Pattern Repository maintains all the AP instances defined on an ESB.

3. After selection, a service executor module instantiates the related services through Service Repository, which is responsible for service abstraction, storage and reconfiguration, and invokes them to process the request.

4. A message may cross several AP containers that are linked by some router and transformation components on ESB. This step is similar with the conventional message process procedures.

5. Finally, a response is sent out to reply the request message. To provide relatively independent services for workflow layer, the ESB should manage some composition services with mediation flows. Moreover, ESB provides service containers that can host multiple services in a black box. PBDR follows the above two principles and provide such a more controllable and flexible routing solution.

C. Dynamic routing in ESB [5] [9] [10]

ESB provides the services for messaging transformation, asynchronous message passing, and message routing. The generic ESB architecture is made up of four parts:

- Message mechanism provides transparent communication services to the applications.
- Message transformation translates the messages of various formats and contents. It hides the heterogeneous in the messages and protocols.
- Routing mechanism establishes the service connections

according to the workflow and navigates the messages from one to the next.

• Container hosts services and provides support for selective deployment, service invocation and lifecycle management.

The key for DRESR to support dynamic routing is to separate service specification from service implementation in the routing path definition. DRESR supports service routing from three perspectives: routing path abstraction with ARP (Abstract Routing Path), instantiation with IRP (Instantiated Routing Path) and reconfiguration. An ARP is a description of the routing path in terms of abstract service names. An IRP maps the abstract service name into the URI of service provider. Reconfiguration reflects the changes in business process and service process. Fig. 3 gives an overall view of the DRESR approach.

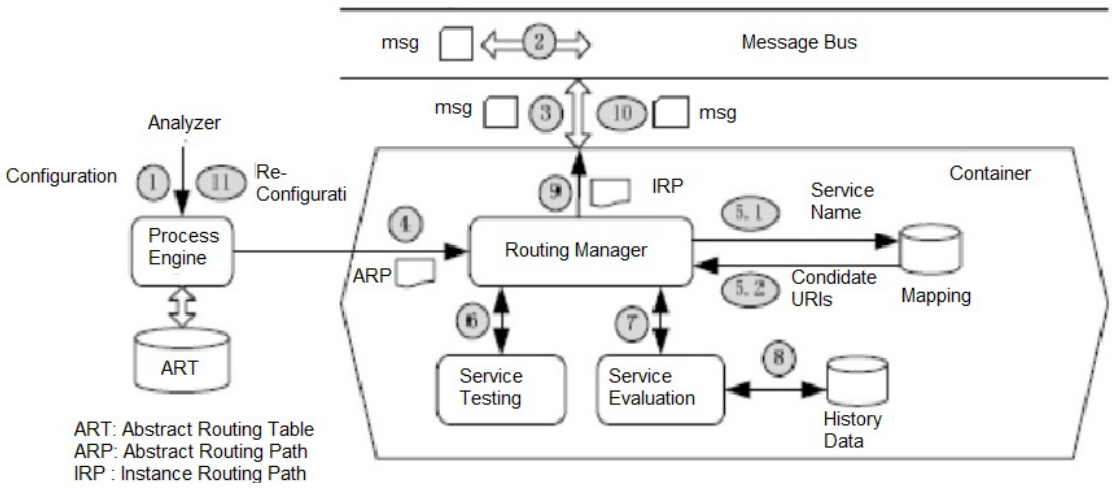


Fig. 3 Dynamic routing in ESB

D.Multifactor-Driven Hierarchical Routing

Multifactor-Driven Hierarchical Routing is a hierarchical routing model to synthesize the PBDR routing solution and layer to support enterprise integration patterns (EIPs). As shown in Fig.3, a Message Filter is configured in MDHR to do a message sorting for the routing modules that have three routing levels: basic level; application level and business level.

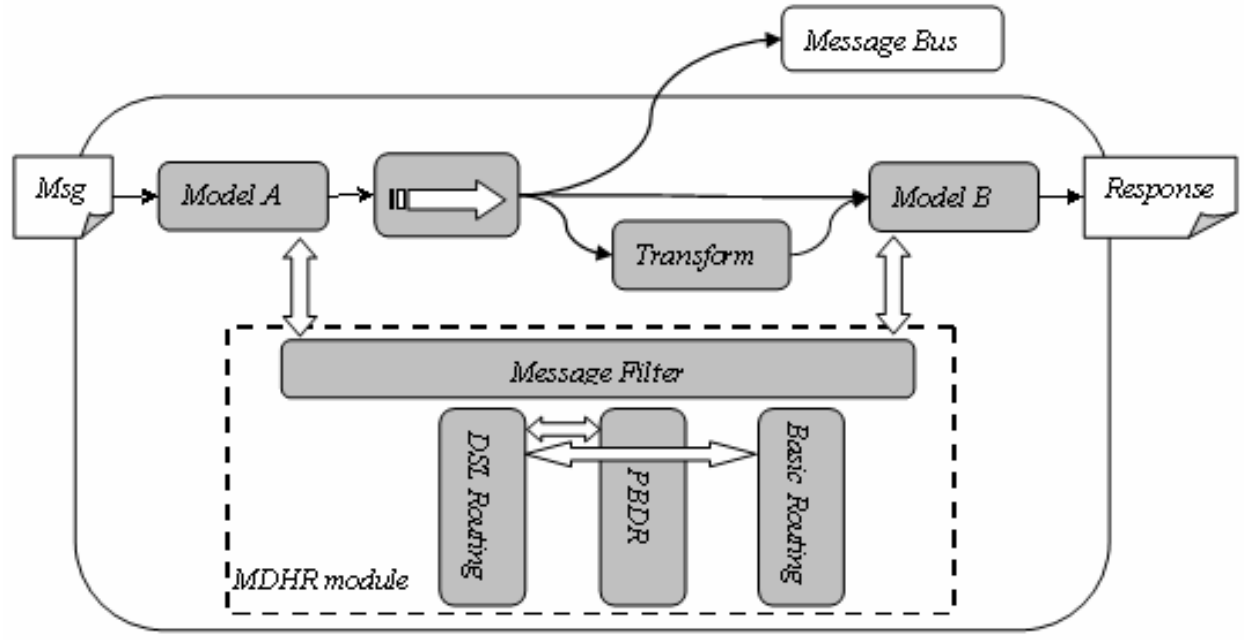


Fig. 4 The MDHR module

introduce a domain specific language (DSL) on top of PBDR

3. Problems of current routing solutions in ESB

Message routing is a very important service communication mechanism. The service requester should not be care which service provider can receive his messages and respond to him .In such condition the requester don't care about service provider anymore and the loose coupling about SOA substantially implemented. So, a kind of tool software to make this function work for dynamic service evocation in the SOA is needed. But the challenge is that how to decide which service provider can be used; is the key problem for it. With the development of the SOA, there are many middlewares designed to fulfill it. By analyzing current message routing in ESB, it could be considered that most of the existing methods are not support dynamic message routing in efficient way. In the real world we need a mechanism to find the service without know from where it comes dynamically. It means that, to complete a business, it is necessary to handle all locations of the valid service providers and access manner at first

4. Intelligent Conceptual Message Routing

Intelligent Conceptual Message Routing (ICMR) is based on tree structure and it illuminated in Fig. 5

E. Service Registration Unit (SRU)

This component is used for registering published services. When a service provider publishes new service, this component is responsible for getting all information about the service and the functionality of it.

F. Service Location Zone UNIT (SLZU)

This component is responsible for analyzing the content and the functionality of the resisted services and the assigning an ID to it. The ID is a unique number that shows the zone of the service in the tree. The second application of this unit is to check whether the service is available or not.

G.Service Delivery Unit (SDU)

This unit is responsible for getting the request from the service consumer and sends it into Service Explorer Unit (SEU) in order to find the appropriate service and to deliver it into consumer.

H.Service Explorer Unit (SEU)

This unit is playing great role in this model. The main functionality of this unit is to get the properties of the required service and find the suitable service according to the requested properties. In order to carry out this task, the SEU using intelligent algorithms to find the service and deliver it. SDU has a Message Agent Module (MAM) which is responsible for ranking the service base on the request and the quality. When one service is requested frequently, the MAM increase its ranking and it will be candidate in next service discovery at the same condition. MAM is using AI algorithms such as Genetic Algorithms to do the ranking and service delivering. In this way the service delivery will be delivered at the shortest time for the frequent requesting service.

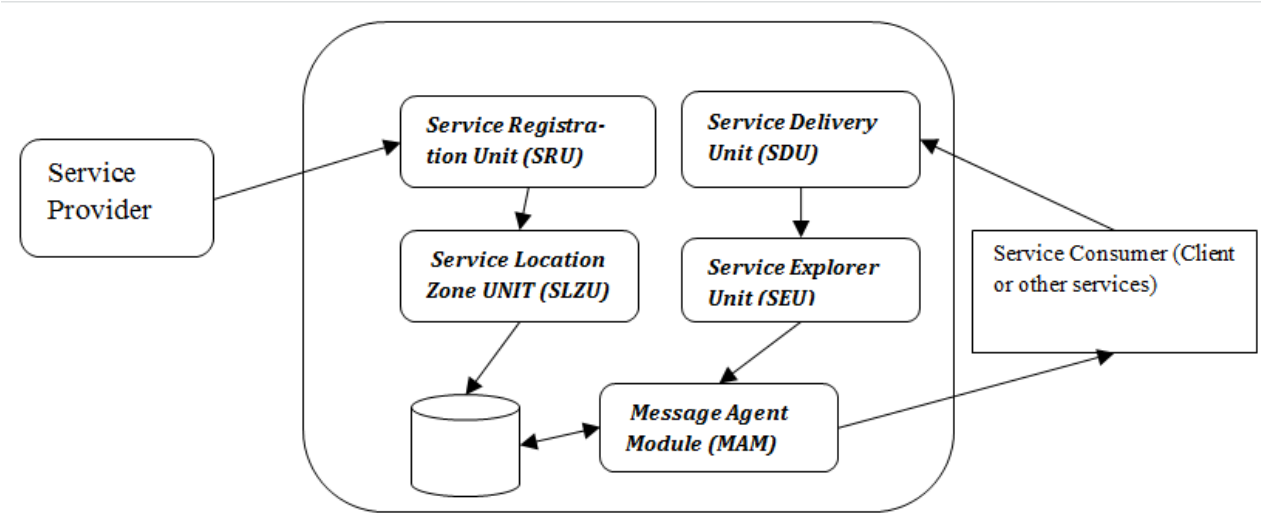


Fig. 5 Intelligent conceptual message routing

5. ICMR Vs. other routing methods

TABLE I shows the differences between our methods and other Message routing methods

TABLE I
ICMR VS. OTHER ROUTING METHODS

Routing Method	Advantage	Disadvantage
Content-Based Routing	• Dynamic routing in run time • Routing based on content of the message [5] • parallel processing of messages at multiple nodes at the same time	Routing is based on prediction Depends on the number of criteria such as the message type, specific field value, existence of field and so on. [5] It is not yet possible (i) to define SOAP message routes with alternative or parallel processing of messages at multiple nodes at the same time. Additionally, (ii) the sequence of message processing by additional services at intermediaries cannot be predefined, even though it is required for the correct execution of applications in multiple scenarios. (iii) The fact that message routes are predefined and allow no parallelism and branching results also in fault-intolerance, since reaction to failures at nodes on the message path is not supported. [11]
Dynamic routing in ESB	• Dynamic reconfiguration at run time by user and systems [5]	• Based on patterns • The invoked services are not guarantee to be alive [10]
Multifactor-Driven Hierarchical Routing	• Multifactor-driven architecture separates the routing configuration from the application arrangement and enables a more adaptable solution when building the service mediation on ESB. [5] • An important functionality of PBDR is the wide-ranging technologies support. [5]	If developers or deployment managers want to add some model on routing layer, they must turn to hard-code or static configuration that will make the ESB too complex to maintain. [5]
Intelligent Conceptual Message Routing	• Users and applications can communicate with the server provider and invoke the desired services by describing their needs. • Reliable service delivery(the dead services will not be delivered) • Search service in efficient way(i.e. searching by Intelligent algorithms) • Search the requested service based on semantics parameters	• Hard to implement some parts of the components(e.g. MAM and service exploring)

6. Conclusion and future works

This paper proposes a hierarchical routing mechanism with several layers to enhance the routing functionality for ESB. Service Registration Unit (SRU) provides several metadata fields for registering the service to mature routing implementations. Service Location Identification Unit (SLIU) is using a Conceptual data structure to arrange service in the hierarchical structure by assigning a unique ID to each service. Service Delivery Unit (SDU) introduces the unit for getting the message from application and delivers the requested service. Through this Intelligent Conceptual model, Applications can be request Service just by defining their needs ESB can be deliver the appropriate service without low performance solutions, like BPEL related system. The Future work includes 1) service mapping and selection algorithm design; 2) the integration with service testing tools based on Symantec service; 3) ESB performance analysis of the newly introduced mechanism.

7. References

[1] G. LI, V. MUTHUSAMY, H. JACOBSEN, *A Distributed Service-Oriented Architecture for Business Process Execution*. ACM Transactions on The Web, Vol. 4, No. 1, Article 2, Publication date: January, 2010.

[2] L. Garc´es-Erice, *Building an Enterprise Service Bus for Real-Time SOA: A Messaging Middleware Stack*. 33rd Annual IEEE International Computer Software and Applications Conference, 2009.

[3] N. M. Josuttis, *SOA in Practice*. O'Reilly Media, Inc, August 2007: First Edition. 978-0-596-52955-0.

[4] M. P. Papazoglou, Heuvel, W. J. Van den, *Service oriented architectures approaches, technologies and research issues*. Springer-Verlag, 2007.

[5] M. Xueqiang, T.Xinhuai, Y.Xiaozhhou, C. Delai, L.Xiangfeng, *Multifactor-Driven Hierarchical Routing on Enterprise Service Bus*. Springer, 2009.

[6] Z.Gulnoza, C. Eunmmi, M. Dugki, *Content-Based Intelligent Routing and Message Processing in Enterprise Service Bus*. International Conference on Convergence and Hybrid Information Technology, 2008.

[7] G. Cugola, E. Di Nitto, *On adopting Content-Based Routing in service-oriented architectures*. Elsevier B.V., 2007.

[8] *Enterprise Integration Patterns*. [Online] http://www.enterpriseintegrationpatterns.com/.

[9] B. Xiaoying, X. Jihui, C. Bin, X. Sinan, *DRESR:Dynamic Routing in Enterprise Service Bus*. IEEE International Conference on e-Business Engineering, 2007.

[10] B. Wu, S. Liu,L. Wu, *Dynamic Reliable Service Routing in Enterprise Service Bus*. IEEE Asia-Pacific Services Computing Conference, 2008.

[11] T. Scheibler, D. Karastoyanova, F. Leymann, *Dynamic Message Routing Using Processes*. [book auth.] Klaus David and Kurt Geihs. Kommunikation in Verteilten Systemen (KiVS). Springer Berlin Heidelberg, 2010, pp. 117-128.

[12] Red Hat. http://www.redhat.com/docs/manuals/jboss/jboss-soa-4.2/html-single/ [Online]
[13] A. Khalili, S. Mohammadi, *Using Logically Hierarchical Meta Web Services to Support Accountability in Mashup Services*. IEEE Asia-Pacific Services Computing Conference, 2008.
[14] W.Hui, F. Peipei, Research of Strategic Route In Heterogeneous System Integration Based On ESB-SOA. IEEE, 2009.
[15] D.Kun,D. Bo,Z. Xiaoyi,L. Ge, *Optimization of Service Selection Algorithm for Complex Event Processing in Enterprise Service Bus Platform.* Proceedings of 2009 4th International Conference on Computer Science & Education, 2009.

O2OOD: A Methodology for Converting Objective-Based Requirements into Object-Oriented Design

Manu Goel
Senior Project Manager, Infosys Technologies Limited,
Plot no.1, Rajiv Gandhi Chandigarh Technology Park,
Kishangarh (Near Mani Majra), Chandigarh, India 160101

Abstract - *Besides providing the much needed clarity to all the stakeholders, Objective-based requirements also provide a smooth transition from requirements stage to (object-oriented) design stage. Through an easy process and slight practice, the objects can be easily discovered from the objective-based requirements. This process involves mapping the objectives to various elements of objects (as far as possible). Here, the idea is to keep the focus on objectives when creating objects. Since the transition from requirements to design is made easier and faster through this conversion process, the obvious benefits are realized from it (saves time, effort and money). This insightful paper primarily proposes a quick and easy methodology (O2OOD) for converting objective-based requirements into object-oriented design.*

Keywords: Object-oriented design, Objective-based requirements, software engineering, software requirements, software design

1 Introduction

Absence of a robust requirements process has been the major contributor towards making 66% software projects a failure or challenged in some way [1]. This has led to people across the industry look for new methodologies in the requirements engineering space. Objective-based requirements are the latest attempt in making the requirements process more robust than ever. Objective Linked Approach for Requirements Development (OLAR) [5] is a good example in this regard. Besides the efforts on bettering the requirements process; there is also a continuous focus on facilitating the expression of requirements and enabling the transformation of requirements into design (especially object oriented design). RUP [2], UML [3] [4] and requirements management [7]; all point towards increased focus on requirements and design processes. O2OOD (Objective to Object-Oriented-Design) methodology is an effort towards enabling the identification of objects/ classes and their relationships when using objective-based requirements process.

2 O2OOD Methodology

O2OOD methodology works in tandem with any objective-based requirements process to pave the path to object oriented design through helping in identification of objects/ classes and their relationships. This methodology involves the use of two matrices to move from requirements to objects/ classes. The entry criteria for using this methodology is: The objectives should have been broken down to their smallest level (e.g. as in OLAR [5]) and the relationships between them established; at least 70% relationships should have been drawn. The exit point is final refinement of objectives and clear identification of objects/ classes and their relationships. The two matrices that are used as part of O2OOD methodology have been shown in figure 1 and figure 2.

Note that the non-functional requirements/ objectives will cut across the complete set of objectives and will have components tied in to (functional) objectives wherever required; however, some non-functional requirements will just act as influencers on the system architecture and hardware/software decisions.

2.1 Objective-Function-Data Matrix

The various elements of the Objective-Function-Data matrix (OFD matrix) have been shown and described in figure 1. Once the project team has synthesized the objectives (and merged them wherever required) to reach the lowest level objectives, they can start using the OFD matrix to gather detailed information about the functions and data that are needed for fulfillment of each of the objective. That said, a lot of times it might be useful to start using the OFD matrix a bit earlier since it can ease the process of splitting and merging of objectives to reach the final set of low level objectives.

Objective Ref no.: <***provide the unique ref no. for the objective*** *e.g. OBJ0033* > **Objective Name:** < ***easily identifiable objective name*** *e.g. calculate interest amount*>	
Algorithmic - Functions	***The description and details of the functions that together form the algorithm required to be followed in order to ensure the fulfillment of this objective.*** *This will actually list out the requirement that is linked to this objective e.g. in the example here (calculate interest amount), the algorithmic- functions will include the algorithm and all the operations (except data functions) that are required to be performed in order to calculate interest amount.*
Data-Functions	***The description and details of the functions that work on data elements (validate/manipulate etc) and hence assist the algorithmic functions in order to ensure the fulfillment of this objective.*** *All data validations, manipulations and operations will come here.*
Data	***The data that is required for the execution of the functions and operations identified above.***
Linked functional objectives	***The reference number of other functional- requirements related objectives that have some common- functionality (or similar functionality) with the functionality contained within this objective.***
Linked non-functional objectives	***The reference number of other non- functional-requirements related objectives that might have an influence on the fulfillment of this objective.***
Other remarks	***Mainly used to specify things/items that need to be looked into later. Also, used for any other comments or remarks.***

Figure 1: Objective-Function-Data Matrix

So, there is an OFD matrix prepared for each objective. Again, the shaping up of the OFD matrix will happen over a series of iterations. So, in the first iteration the project team will be able to come up with only algorithmic functions, data functions, data and some of the linked functional objectives. In the second iteration, project team will be able to add the non-functional objective dependencies/ considerations to the OFD matrix. Also, from second iteration onwards the project team might find it useful to start looking into the OFD matrices of the linked function objectives. At this stage, the Objective-Refine (OR) matrix can be started too.

2.2 Objective-Refine (OR) matrix

This matrix has been shown in figure 2 and the various elements have been explained in the figure itself. The OR matrix can be started as soon as the OFD matrices are ready for most of the objectives. The prime motive of this matrix is to use the information gathered through OFD

matrices and represent them in such a way that merging/ splitting of objectives can be facilitated and at the same time a clear cut view of the possible objects/ classes is made available for the design team to use. This matrix makes use of the objective-relationships that have been established in the OFD matrices.

All the functions and data for the related (linked) objectives is filled in the OR matrix. It would be preferred that OR matrix is prepared in a spread sheet kind of format where sorting/ filtering functionality (for various columns of the matrix) is available on the click of a button. In order to make things better organized, it is suggested that instead of an OR matrix, a set of OR matrices is prepared (one each for a set of related objectives).

Once the OR has been filled with the information from individual OFD matrices, it is time to start the analysis and decide on split and merge. For this, just use a filter in the spreadsheet (containing the OR matrix) to look at the algorithmic functions first. If there are similar functions across objectives, consider creating a new objective (and in turn a new OFD together with modified OFDs) that would have that function or functions (and the related data, if any) that would then be linked to all those objectives that need that function – this is a classic example of a split. Similar exercise needs to be done for data functions and data too. Across iterations, the objectives will get refined and the relationships between various objectives will be very clear.

At this stage, the OR matrix will contain a list of functions and data that can easily be transformed into classes and objects for creating an object oriented design. Mostly, the data functions and the data they manipulate will lead to entity classes and the algorithmic functions will lead to control classes (and boundary classes). Since the segregation of objectives has already been done at a low level, the classes/objects so obtained will be in an almost-final state and can be quickly used to create the object oriented design. The segregation of algorithmic functions will also enable easy application of design patterns because the objectives of the classes that contain such algorithm will be clearer than what it is when following the usual process for object identification.

SN.	Obj Ref.	Function	Algo/ Data?	Data	Split/ Merge?	Remark

- **SN.** = Serial number
- **Obj Ref.** = Objective reference number (unique)
- **Function** = Very short description of the data and algorithmic functions
- **Algo/Data?** = The function type (whether algorithmic or data).
- **Data** = Details of data entities involved for that function
- **Split/ Merge**? = Whether the function needs to be split from the current objective (and be created as part of a new objective) or whether it can be merged with an existing objective. If the function can be merged with an existing objective, the objective reference number should be provided in the 'Remark' column.
- **Remark** = Comments /remarks for the line items.

Figure 2: Objective-Refine Matrix

3 Discussion

The O2OOD methodology provides the much needed direction on easy movement of requirements into design. Though O2OOD outlines the matrices and their formats and the process that the project teams need to follow, the project teams are encouraged to brainstorm and tailor the matrices as per their needs. Similarly, it is expected that the project team will brainstorm on the nomenclature to be used throughout the adoption of O2OOD methodology into the project. Again, the number of iterations required will be based on how much refinement (of objectives) happens in each iteration. It is suggested that initial iterations are done pretty much in a workshop mode till the time first set of OR matrices is created. This will help iron out team synchronization and communication issues.

As such O2OOD is expected to bring a number of benefits to the project in which it is adopted. One of the prime benefits is the easy recognition of objects and classes; and hence a lot of time, effort and money is saved. However, another inherent benefit is 'enhanced understanding of the requirements'. As project team iterates through the process of refining OFDs and ORs, they get a better understanding of the requirements, their objectives and how they link together (and across); this enables their decision making, helps them make right assumptions and establishes a better communication with the business team. This, in turn, leads to enhanced bonding between various stakeholders and quick consensus on what would otherwise have been contentious items. Further, it enhances the confidence levels of all stakeholders involved.

4 Conclusions

Any system or application or IT project is built for fulfillment of an objective or a set of objectives. In that sense it makes a lot of sense to keep the objectives at the center of the whole process of creating the system / software etc. Such focus on objectives can not only help in getting the right set of requirements out, but also help all the stakeholders in getting a better understanding of the requirements. So, the first part is using an objective-based requirements process to gather requirements.

And the second part is further enhancing the process to facilitate the linking between the requirements stage and the design stage of an IT project. Considering the fact that the requirements and design stage defects are 100 times costlier to find and correct [6], such linking between requirements and design becomes even more important. The O2OOD methodology provides a direction in this regard and is expected to be useful for most IT development projects.

5 References

[1] Standish Group CHAOS Report shows success rates have improved by 50%, *Business Wire*, 2003.

[2] Philippe Kruchten, *Rational Unified Process - An introduction*, Addition-Wesley, 1999

[3] Terry Quatrani, *Visual modeling with rational rose 2002 and UML*

[4] Dan Pilone and Neil Pitman, *UML 2.0 In a Nutshell*, O'Reilly, 2005

[5] Manu Goel, *"*Are you plagued with unnecessary costs in your IT projects*"*, *SetLabs Briefings*, Vol 6 No 4, 2008

[6] B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer, IEEE Computer Society*, Vol. 34, No. 1, January 2001

[7] Karl E. Wiegers, *Software Requirements*, WA: Microsoft Press, Redmond, Second Edition 2003.

Recovering Activity Diagrams from Object Oriented Code: an MDA-based Approach

L. Martinez [1], C. Pereira [1] and L. Favre[1,2]
[1]Universidad Nacional del Centro de la Provincia de Buenos Aires
Campus Universitario, Paraje Arroyo Seco, Tandil, Argentina
[2]Comisión de Investigaciones Científicas de la Provincia de Buenos Aires, Argentina

Abstract - *The success of system modernization depends on the existence of technical frameworks for information integration and tool interoperation like the Model Driven Architecture (MDA). Reverse engineering techniques play a crucial role in system modernization. This paper describes how to reverse engineering activity diagrams from object oriented code in the MDA context focusing on transformations at model and metamodel levels. A framework to reverse engineering MDA models from object oriented code that distinguishes three different abstraction levels linked to models, metamodels and formal specifications, is described. At model level, transformations are based on static and dynamic analysis. At metamodel level, transformations are specified as OCL contracts between MOF (Meta Object Facility) metamodels which control the consistency of these transformations. The level of formal specification includes algebraic specifications of MOF metamodels and metamodel-based transformations. This paper analyzes a recovery process of activity diagrams from Java code by applying static and dynamic analysis and show a formalization of this process in terms of MOF metamodels.*

Keywords: Metamodeling, Meta Model Facility, Reverse Engineering, Model Driven Architecture, Transformation

1 Introduction

Reverse engineering is the process of analyzing available software artifacts such as requirements, design, architectures, code or byte-code with the objective of extracting information and providing high-level views on the underlying system [16].

At the beginning, reverse engineering was focused mainly on recovering high-level architecture or diagrams from procedural code to deal with problems such as comprehending data structures or databases. At that time, many different kinds of static analysis techniques, basically based on compiler theory and abstract interpretation, were developed.

Later, when object oriented languages emerged, a growing demand of reverse engineering systems appeared and the focus of software analysis moved from static to dynamic analysis. With the emergency of the Unified Modeling Language (UML) [18][19], reverse engineering was focused on how to extract higher level views of the system expressed by different kinds of diagrams.

Nowadays, software and system engineering industry evolves to manage new platform technologies, design techniques and processes [8]. New technical frameworks for information integration and tool interoperation such as the Model Driven Development (MDD) have created the need to develop new analysis tools and specific techniques. MDD refers to a range of development approaches that are based on the use of software models as first class entities, being one of them the Model Driven Architecture (MDA) [12]. MDA distinguishes different kinds of models: computation independent model (CIM), platform independent model (PIM), platform specific model (PSM) and implementation specific model (ISM). In MDA, artifacts generated during software development are represented using common metamodeling languages. The essence of MDA is MOF metamodel that allows different kinds of artifacts from multiple vendors to be used together in the same project [13].

With the emergence of MDA, new approaches should be developed to reverse engineering PIMs and PSMs from object oriented code. This paper contributes a framework to reverse engineering MDA models from object oriented code based on the integration of static and dynamic analysis techniques, metamodeling and formal specification. In a previous work we show how to reverse engineering PSMs including class diagrams and state diagrams [9]. Nowadays, we are working in reverse engineering PIMs such us activity diagrams and use cases. In this paper, the emphasis is given to reverse engineering activity diagrams from Java code focusing on static analysis. Also, we show how to integrate these results with dynamic analysis. MOF-based specifications are used to analyze the consistency of model recovery processes.

This paper is organized as follows. Section 2 deals with related work. Section 3 describes a framework for MDA-based reverse engineering. Section 4 describes code-to-model transformation based on static analysis. Also, it shows how to recover activity diagrams from Java code. Section 5 specifies reverse engineering process as metamodel-based transformations. Finally, Section 6 highlights conclusions.

2 Related Work

Many works have contributed to reverse engineering object oriented code. An overview of the state-of-the-art of reverse engineering techniques is presented in [2]. Reference

[5] compares existing works work in the area of reverse engineering, discusses success and provides a road map for possible future developments in the area. Reference [17] provides an overview of techniques applied in the field of reverse engineering of object oriented code.

With the emergence of MDA, new approaches have been developed. An approach for bridging legacy systems to MDA is presented in [15]. A framework for automatic legacy system migration in MDA is described in [4]. The article [11] reports on a project that assessed the feasibility of applying MDD to the evolution of a legacy system. The article [9] describes a framework and exemplifies the reverse engineering of class diagrams and state diagrams from object oriented code.

Although many Case tools support reverse engineering, they provide little support for validating models and transformations. Commercial tools do not support reverse engineering of PIM, use cases and activity diagrams in particular [5] [6].

Our contribution complements the reverse engineering process described in [17]. We propose a formalization of the recovery processes in terms of standards involved in MDA such as MOF metamodels and, we analyze how to recover PIMs, activity diagrams in particular. Also, we contribute a process that allows extending the functionality of the existing MDA Case tools.

3 MDA-Based Reverse Engineering

To reverse engineering MDA models from object oriented code, we propose a framework based on the integration of compiler techniques, metamodeling and formal specification. This framework distinguishes three different abstraction levels linked to models, metamodels and formal specifications (Fig.1).

The model level includes code, PIMs and PSMs. PIMs and PSMs are expressed by means of UML class diagrams annotated with OCL [14].

At model level, transformations are based on classical compiler construction techniques. They involve processes with different degrees of automation, which can go from totally automatic static analysis to processes that require human intervention, to dynamically analyze the resultant models. All the algorithms that deal with reverse engineering share an analysis framework. The basic idea is to describe source code or models by an abstract language and perform a propagation analysis in a data-flow graph called in this context object-data flow. This static analysis is complemented with dynamic analysis supported by tracer tools.

The metamodel level includes MOF metamodels that describe the transformations at model level. MOF metamodel uses an object modeling framework that is essentially a subset of UML core [18]. The modeling concepts are classes, associations, data types and packages. MOF metamodels describe families of ISMs, PSMs and PIMs. Transformations are specified as OCL contracts between a source metamodel and a target metamodel. MOF metamodels "control" the consistency of these transformations.

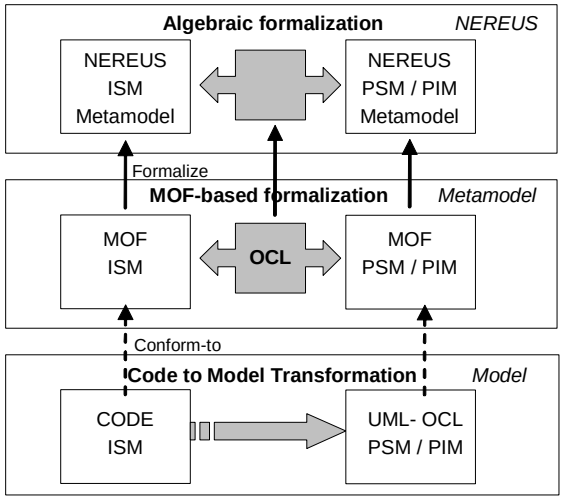


Figure 1. A framework for MDA-based reverse engineering

The level of formal specification includes specifications of MOF metamodels and metamodel-based transformations in the metamodeling language NEREUS that, like MOF, are based on the concepts of entity, association and system. NEREUS can be used as a common specification language and is connected with different formal and programming languages, [7] shows how to integrate NEREUS with the Common Algebraic Specification Language (CASL) [3].

This paper focuses on reverse engineering of PIMs from object oriented code at model and metamodel level. A process for recovering activity diagrams from Java code at PIM level is analyzed and specified as metamodel-based transformations.

4 Code to Model Transformation

At model level, transformations are based on static and dynamic analysis. Static analysis extracts static information that describes the structure of the software reflected in the source code whereas dynamic analysis information describes the structure of the run-behavior. Static information can be extracted by using techniques and tools based on compiler techniques such as parsing and data flow algorithms [1]. Dynamic information can be extracted by using debuggers, event recorders and general tracer tools.

An MDA-based reverse engineering process abstracts MDA models from ISMs. This process involves different phases. First, the source code is parsed to obtain an abstract syntax tree (AST) associated with the source programming language grammar. Next, a metamodel extractor extracts a simplified, abstract version of a language that ignores all instructions that do not affect the data flows, for instance all control flows such as conditionals and loops.

The information represented according to this metamodel allows building the data-flow graph for a given source code, as well as conducting all other analysis that do not depend on the graph. The idea is to derive statically information by performing a propagation of data. Different kinds of analysis propagate different kinds of information in the data-flow graph extracting the different kinds of diagrams.

The static analysis is based on classical compiler techniques [1] and abstract interpretation [10]. On the one hand, data- flow graph and the generic flow propagation algorithms are specializations of classical flow analysis techniques [1]. On the other hand, abstract interpretation allows obtaining automatically as much information as possible about program executions without having to run the program on all input data and then ensuring computability or tractability. In reverse engineering process, static analysis can be enriched with dynamic one. Dynamic analysis is based on an execution model generated by execution tracer tools. The execution model includes the following components: a set of objects, a set of attributes for each object, a location and value of an object type for each object, and a set of messages. Additionally, types such as Integer, String, Real and Boolean are available for describing types of attributes and parameters of methods or constructors.

Static and dynamic information could be shown as separated views or merged in a single view. The dynamic behavior could be visualized as an execution scenery which describes interaction between objects. To extract specific information, it is necessary to define particular views of these sceneries.

4.1 Models Recovery

In the first stage of the process of recovering diagrams, the Java language is simplified into an abstract language, where all features related to the object flow are maintained while the other syntactic details are dropped. The choice of this program representation is motivated by the computational complexity and the "nature" of the object oriented programs whose code is typically structured so as to impose more constraints on the data flows than on the control flows. For example, the sequence of method invocations may change when moving from an application which uses a class to another one, while the possible ways to copy and propagate object references remains more stable. Fig. 2.a shows the abstract syntax of the simplified Java language. A Java program P consists of zero or more occurrences of declarations (D) followed by zero or more statements (S). Declarations are of three types: attribute declarations, method declarations and constructor declarations. Statements are of three types: allocation statements, assignment statements and method invocations.

This language is the basis for the definition of the Object Flow Graph (OFG), which is a pair (N, E) where N is a set of nodes and E is a set of edges. A node is added for each program location (i.e., local variable, attribute or formal parameter). Edges represent the data flows appearing in the program. Nodes and edges are constructed according to the rules depict en Fig. 2.b.

When a constructor or method is invoked, edges are built which connect each actual parameter a_i to the respective formal parameter f_i. In case of constructor invocation, the newly created object, referenced by *cs.this* is paired with the left hand side *x* of the related assignment. In case of method invocation, the target object *y* becomes *m.this* inside the called method, generating the edge (*y*, *m.this*), and the value returned by method *m* (if any) flows to the left hand side *x* (pair (*m.return*, *x*)).

Some edges in the OFG may be related to the usage of class library. Each time a library class introduces a data flow from a variable *x* to a variable *y* an edge (*x,y*) must be included in the OFG. Containers are an example of library classes that introduce external data flows. Object containers provide two basic operations affecting the OFG: *insert* and *extract* for adding an object to a container and accessing an object in a container respectively. In the abstract program representation insertion and extraction methods are associated with container objects.

4.2 Activity Diagram Recovery

A reverse engineering approach of object oriented code is described in [17]. Tonella and Potrich [17] adapted concepts and algorithms of data flow analysis described in [1] to obtain UML diagrams from Java code, particularly class diagrams, object diagrams, interaction diagrams, state diagrams and package diagrams. Although the authors do not reverse engineering activity diagrams, this process can take place applying the same principles. In this paper, we complement this work to reverse engineering activity diagrams from Java code in the MDA context.

Activity diagrams model the dynamic aspects of a system. They show the flow from activity to activity within a system. Activity diagrams may stand alone to visualize, specify, construct, and document the dynamics of a society of objects, or they may be used to model the flow of control of an operation [19]. The abstract language obtained in the first stage of the process of recovering diagrams allows recovering activity diagrams to model workflow of an operation. This process makes sense if the operation is complex enough, that is to say, it involves many activities.

The OFG represents all data flows involving objects and allows tracing the flow of information about objects from the object creation by allocation statements, trough object assignment to variables, up until the storage of objects in class fields or their usage in method invocation. A generic flow propagation algorithm working on the OFG is used to infer properties about the program objects.

a. Abstract syntax	b. OFG edges induced
P ::= D * S *	{}
D ::= a	{}
| m (f_1, ..., f_k)	{}
| cs (f_1, ..., f_k)	{}
S ::= x = new c (a_1, ...,a_k);	{ (a_1, f_1) ∈ E, ..., (a_k, f_k) ∈ E, (cs.this, x) ∈ E }
| x = y;	{ (y, x) ∈ E }
| [x =] y.m (a_1, ...,a_k);	{ (y, m.this) ∈ E, (a_1, f_1) ∈ E, ..., (a_k, f_k) ∈ E, (m.return, x) ∈ E }

Figure 2. Java abstract syntax and OFG edges

The following pseudo-code of generic flow propagation algorithm is a specific instance of the algorithms applied to control flow graph described in [1]:

```
for each node n ∈ OFG
        in[n] = ∅
        out[n] = gen[n] U ( in[n] - kill[n] )
end for
while any in[n] or out [n] changes
        for each node n ∈ OFG
                in[n] = U_{p∈pred(n)} out[p]
                out[n] = gen[n] U (in[n] - kill[n])
        end for
end while
```

Each node *n* of the OFG stores the incoming and outgoing flow information respectively inside the sets *in[n]* and *out[n]*, which are initially empty. Moreover, each node *n* generates the set of flow information items contained in the *gen[n]* set, and prevents the elements in the *kill[n]* set from being further propagated after node *n*. Incoming flow information is obtained from the predecessors of node *n* as the union of the respective out sets (forward propagation).

To recovery activity diagrams which model the operation workflow, the following diagram elements are required:

1. *action states*: actions are deduced from method calls. For instance:

```
class A
   f () {      ... p.g();   //   this --- g ---> p  }
```

this, instance of A, sends the message *g* to object *p*. *p* may be a variable of *f* or a class attribute. The message *g* represents an action carried out by *p*.

2. *swimlanes*: they are inferred from the classes that implement the action.

3. *transitions*: they are determined by dynamic analysis.

4. *objects*: they are deduced from local variables and parameters of the operation. These objects are connected using a dependence relationship to the action or transition that creates, destroys or modifies them. This use of dependency relationships and objects is called an object flow, it represents the participation of an object in a control flow.

The main steps to recover activity diagram elements that model the operation workflow are the followings:

1. Apply the algorithm of object flow propagation on the OFG constructed from the abstract code of the operation.

2. Execute the activity resolution algorithm for each call expression in the operation abstract code. Next, the pseudocode of the algorithm is shown:

```
resolveActivity (expr: 'p.g(a1,…,ak )') : Pair
A ← class (scope(expr))
f ← method (scope(expr))
objects ← ∅
if  p is a class attribute
     swimlanes ← types(out [A.p])
     else swimlanes ← types(out [A.f.p])
endif
for each i in  1≤ i ≤ k
     objects  ← objects  ∪ in[A.f.ai] end for
return (swimlanes, objects)
```

resolveActivity is applied to a call expression of the form *p.g(a1,…,ak)* inside a method *f* of class *A*. As a result, this algorithm returns a pair of sets, *swimlanes* and *objects*, containing types and object identifiers respectively. The *swimlanes* set is obtained by the types of the objects referenced by the location *p (out[A.f.p]* or *out[A.p]* in case *p* is a class attribute). More complex Java expressions involving method calls can be easily reduced to the case reported in *resolveActivity*. For instance, if a chain of attribute accesses precedes the method call, as in *p.q.g()*, the invocation targets are obtained from the last involved attribute: *out[B.q]*, where *B* is the class of the attribute *q* accessed through *p*. The *objects* set is obtained from the objects referenced by the location $a_i(in[A.f.a_i])$.

3. Once objects, swimlanes and action states have been obtained, dynamic analysis allows determining the call flow (sequential, concurrent or alternative) through the execution traces generated by executing the code on test cases. The information that must be available from the execution traces to support the construction of activity diagrams consists of:

- object identifiers which are computed within the execution of class constructors;
- identifier of the current object and of the object on which each invocation is issued, which are included to the execution traces;
- time stamps linked with method calls which are produced and traced.

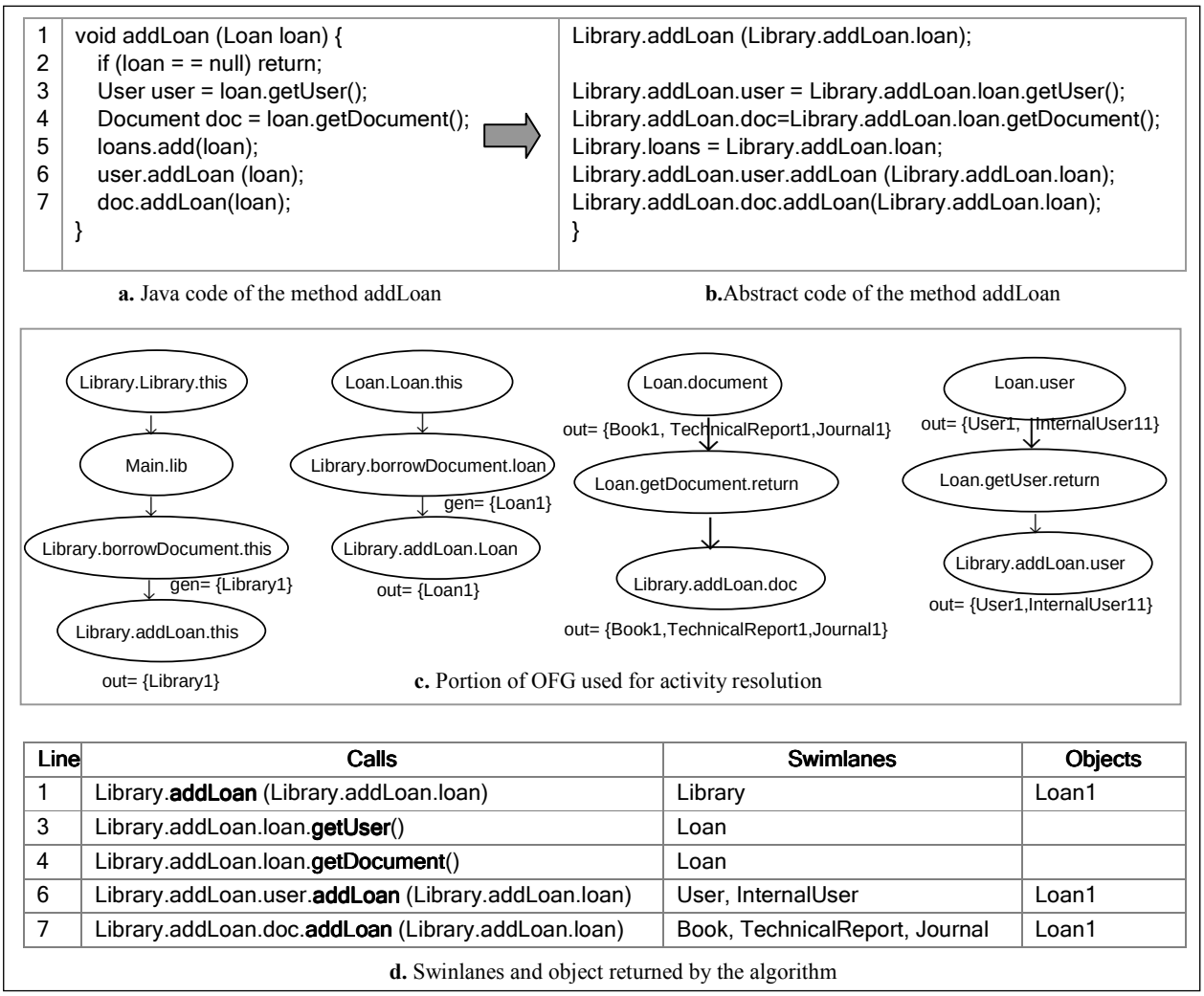
The processing of the execution traces provide an activity diagram for each trace executed and depends on the quality of the test cases. Moreover, dynamic analysis can be used to detect functionality that may never be executed. In this case, the software engineer knowledge and expertise will be required to determine whether it is unreachable code.

Next, the activity diagram is obtained drawing an action for each call expression in the corresponding swimlane. The transitions are drawn among action states from the information obtained through the dynamic analysis. The diagram is complemented with the objects and object flow.

Reference [17] exemplifies reverse engineering by using the Java program *eLib* that supports the main library functions. We exemplify the steps of the recovery process in terms of this example. Fig. 3 exemplifies the approach. Fig. 3.a and 3.b show Java code and abstract code of method *addLoan* of the class Library respectively. Fig. 3.c shows the portion of OFG that contain the information obtained from applying the algorithm of object flow propagation, which is required to apply the activity resolution algorithm. Fig. 3.d shows objects and swimlanes for each method call inside the method *addLoan* which are obtained by the *resolveActivity* algorithm.

Finally, the dynamic analysis returns the transitions between action states. Test cases for the *eLib* program where the parameter of the *addLoan* method is invalid will produce the function finalization. Otherwise, if the parameter is valid, execution traces provide information about action states and the control flow between them.

From this information, it is straightforward to build the activity diagram for the method *addLoan* (Fig.4).



Line	Calls	Swimlanes	Objects
1	Library.**addLoan** (Library.addLoan.loan)	Library	Loan1
3	Library.addLoan.loan.**getUser**()	Loan	
4	Library.addLoan.loan.**getDocument**()	Loan	
6	Library.addLoan.user.**addLoan** (Library.addLoan.loan)	User, InternalUser	Loan1
7	Library.addLoan.doc.**addLoan** (Library.addLoan.loan)	Book, TechnicalReport, Journal	Loan1

d. Swinlanes and object returned by the algorithm

Figure 3. An example: from method addLoan to activity diagram

5 Specifying Reverse Engineering in MDA

A metamodeling technique is used to specify reverse engineering in MDA. MOF metamodels are used to describe the transformations at model level. For each transformation, source and target metamodels are specified. A source metamodel defines the family of source models to which the transformation can be applied. A target metamodel characterizes the generated models.

The transformations between models are described relating each element of the source model to one or more elements of the target model at metamodel level. Transformations, expressed as contracts specified in OCL, consist of parameters, precondition, postcondition and local operations. Each parameter is a metamodel element. The precondition states relations at the metamodel level between the elements of the source model. The postcondition states relations at metamodel level between the elements of the source model and the target model. Local operations are used in preconditions and postconditions.

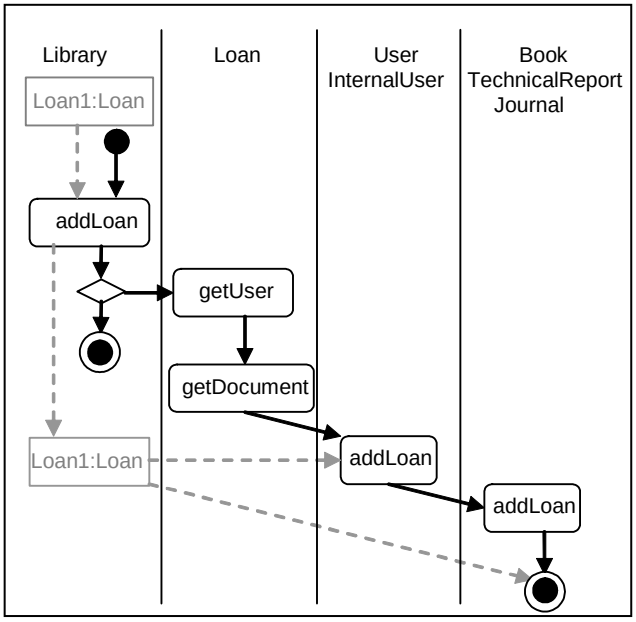


Figure 4. Activity diagram of the method addLoan

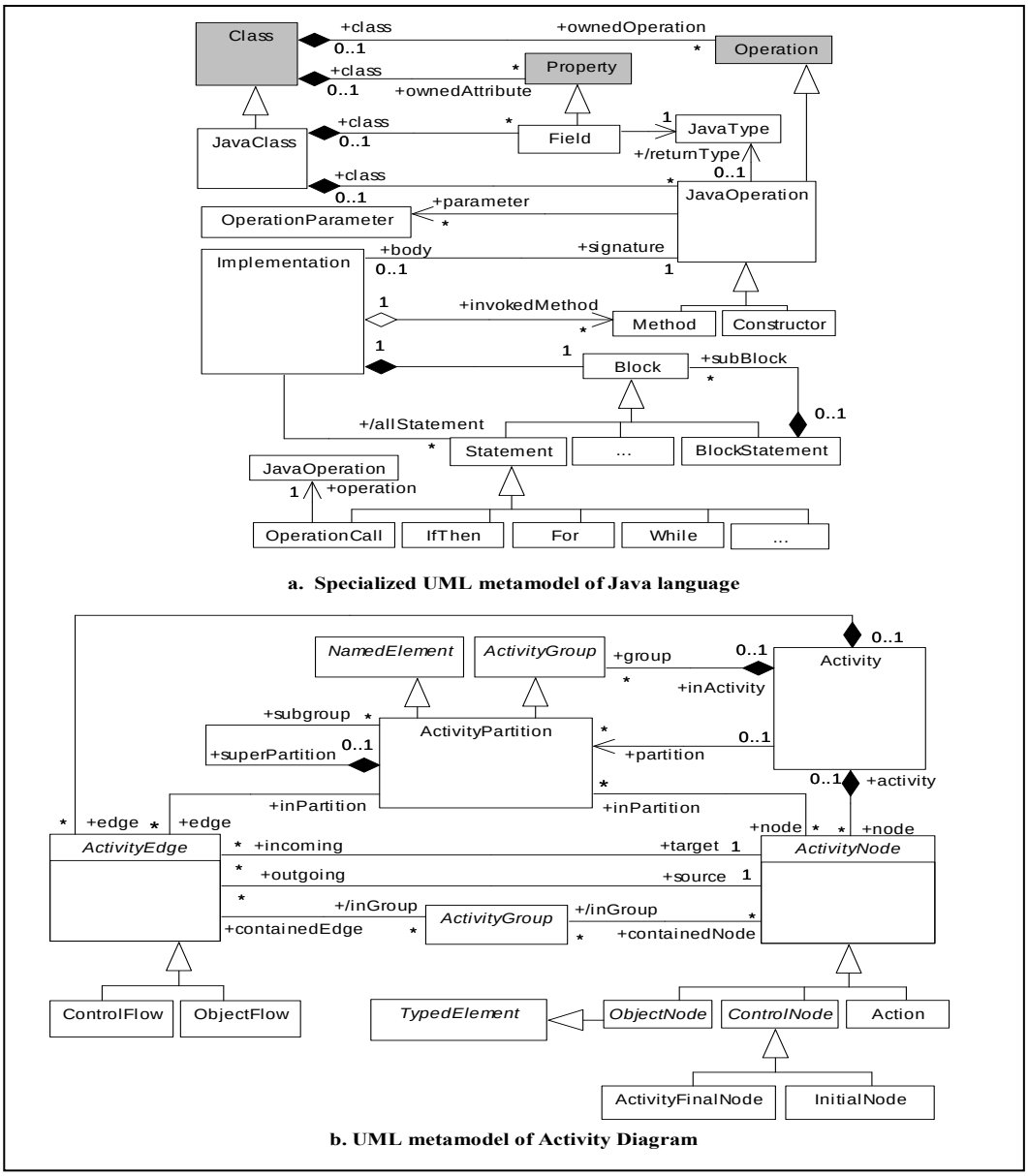


Figure 5. Source and target metamodel

5.1 Specifying Reverse Engineering Activity Diagram

To specify reverse engineering of activity diagrams at metamodel level, source and target metamodels are specified. The source metamodel corresponds to the Java language which is a specialized UML metamodel. Fig. 5.a partially shows this metamodel, the shaded metaclasses correspond to metaclasses of the UML metamodel, whereas the remainder correspond to the specialized metaclasses. It includes constructs that represent Java classes, which own fields and operations. For instance, an operation could have an implementation composed by statements. The target metamodel corresponds to the UML activity diagram, which is partially shown in Fig. 5.b. The metamodel defines a set of concepts that can be used for modeling dynamic aspects of system such as activities, transitions and object flow.

Model transformation is specified relating each element of the source model to one or more elements of the target model at metamodel level. Fig. 6 partially depicts the specification of the transformation as an OCL contract. The rule parameters are metamodel elements, source corresponds to a method of Java-code metamodel and target corresponds to an activity of activity diagram metamodel. The specification is explained by comments.

6 Conclusions

This paper describes a framework to reverse engineering MDA models that is based on the integration of static and dynamic analysis, metamodeling and formal specification. Although, we focus on the static analysis, the paper partially describes how dynamic analysis complements static analysis.

```
Transformation JavaCode-to-ActivityDiagram {
parameter
   source: Java-Code-Metamodel: Method
   target: ActivityDiagram-UML-Metamodel: Activity
local operations
preconditions ...
postconditions
post
-- the first generated node has the name of the 'source' method and
-- contains an incoming edge from an initial node. Each parameters
-- of the method corresponds to an object node.
...
post
--For each sentence Call of the method 'source'
source.implementation.allStatement ->select (sentence|
     sentence.oclIsTypeOf(OperationCall)
   -- there is a node ActivityNode so that
    target.node -> exists ( node    | --sentence corresponds to node
    sentence.oclAsType(OperationCall).opCallMatch(node)))
-- For each sentence IfThen of the method 'source'
source.implementation.allStatement ->select (sentence |
     sentence.oclIsTypeOf(IfThen)
     -- there is a node DecisionNode so that
     target.node -> exists ( node | --sentence corresponds to node
          sentence.oclAsType(IfThen).conditionalMatch(node) ))
...
local operations

Java-Code-Metamodel::OperationCall :: opCallMatch (
  node : ActivityDiagram-UML-Metamodel::ActivityNode ): Boolean

opCallMatch (node) =
-- opCallMatch verifies that a call sentence in source corresponds
-- to an activity node in target by checking the following properties:
   -- the type of node is Action ,
   node.oclIsTypeOf(Action)  and
   -- node and 'self' have the same name
   node.name = self.operation.name and
   -- node is contained in a partition whose name is the same as
   -- the class that owns 'self'
   node.inPartition.name = self.operation.class  and
      -- the number of the incoming edges is equal to the number
      --of 'self' parameters
      node.incoming ->select (inc | inc.oclIsTypeOf(ObjectFlow))
           ->size() = self.operation.parameter -> size()  and  ...
}
```

Figure 6. JavaCode-to-ActivityDiagram transformation in OCL

The main contribution of this paper is a recovery process of activity diagram from Java code by applying static analysis and its formalization in terms of transformations based on MOF metamodels. These transformations, specified as OCL contracts, allow analyzing the consistency of model recovery processes.

These results are integrated with previous ones that describe reverse engineering of PSMs including class diagrams, state diagrams and use cases.

7 References

[1] Aho, A., Sethi, R., and Ullman, J. 1985. "Compilers: Principles, Techniques, and Tools" (2nd ed) Addison-Wesley.

[2] Angyal, L., Lengyel, L., and Charaf, H. 2006. "An Overview of the State-of-the-Art Reverse Engineering Techniques". 7th International Symposium of Hungarian Researchers on Computational Intelligence. 507-516.

[3] Bidoit, M. and Mosses, P. 2002. "CASL User Manual-Introduction to Using the Common Algebraic Specification Language". LNCS 2900. Springer-Verlag

[4] Boronat, A., Carsi, J., and Ramos, I. 2005. "Automatic reengineering in MDA using rewriting logic a transformation engine". 9th European Conf. on Software Maintenance and Reengineering. CSMR´05. IEEE Computer Society. 228-231.

[5] Canfora, G. and Di Penta, M. 2007. "New Frontiers of reverse Engineering". Future of Software engineering. Future of Software Engineering. FOSE 2007. IEEE Press. 326-341.

[6] CASE TOOLS 2010. www.objectbydesign.com/tools

[7] Favre, L. 2009. "A Formal Foundation for Metamodeling. Reliable Software Technologies" ADA Europe 2009. LNCS 5570. Springer-Verlag. 177-191.

[8] Favre, L. 2010. "Model Driven Architecture for Reverse Engineering Technologies: Strategic Directions and System Evolution". Engineering Science Reference, 1 Edition. USA.

[9] Favre, L., Martinez, L., and Pereira, C. 2009. "MDA-based Reverse Engineering of Object-Oriented Code". Lecture Notes in Business Information Processing, 29. Heidelberg: Springer-Verlag. 251-263.

[10] Jones, N. and Nielson, F. 1995. "Abstract interpretation: A semantic based tool for program analysis". Handbook of Logic in Computer Science, Vol. 4. Gabbay D., Abramsky, S., Maibaum, T. Eds. Clarendon Press, Oxford. 527–636.

[11] MacDonald, A., Russell, D., and Atchison, B. 2005. "Model driven Development within a Legacy System: An industry experience report". 2005 Australian Software Engineering Conference. ASWEC 05. IEEE Press. 14-22.

[12] MDA 2011. www.omg.org/mda

[13] MOF 2006. OMG: formal/2006-01-01.

[14] OCL 2010. OMG: formal/2010-02-01.

[15] Qiao, B., Yang, H., Chu, W., and Xu, B. 2003. "Bridging legacy systems to model driven architecture". 27th Annual International Computer Aided Software and Applications Conference. IEEE Press. 304-309.

[16] Sommerville, I. 2004. "Software Engineering" (7th ed.). Addison Wesley.

[17] Tonella, P. and Potrich, A. 2005. "Reverse Engineering of Object Oriented Code". Monographs in Computer Science. Heidelberg: Springer-Verlag.

[18] UML Infrastructure 2010. OMG: formal/ 2010-05-03.

[19] UML Superstructure 2010. OMG: formal/2010-05-05.

Harris Hierarchy of Software Development Support Needs

Michael Harris and Thomas M. Cagley, Jr.
David Consulting Group, Malvern, PA, USA

Abstract – *This paper describes the authors' hypothesis that software development organizations operate at different levels in a hierarchy of needs modeled on the Maslow Hierarchy of needs. The Harris Hierarchy of Software Development Needs has four levels representing operating states of the software development organization: Failure, Fear of Failure, Control and Value. The paper describes how the new model can be applied to software development organizations. It describes what an assessment might look like for an example organization with an actual assessment for a real organization. The authors suggest metrics for the assessment. Finally, the paper suggests events that might cause a positive or negative transition through the level of the model.*

Keywords: Model based software engineering, Software Engineering Methodologies, Project Management Issues, Software Metrics

1 Introduction

This research paper describes a hypothesis that emerged, partially formed, from an internal research seminar conducted by the leading consultants of the David Consulting Group as part of continuing efforts to improve the software development of clients. The genesis of this particular hypothesis was an attempt to synthesize our many years of experiences with software development organizations into a set of categories that would enable us to better help existing and future clients to recognize their current and desired situation.

We have tested the hypothesis against our own experiences and found it to be a useful tool in most cases but the hypothesis needs to be validated against the experiences of others in the community. We will welcome feedback.

2 Harris Hierarchy of Software Development Needs

Recently, we were discussing our clientsô reasons for needing help with their software development functions. As the ñneedò stories and examples flew, we found the same common thread kept coming up again and again: Failure, Fear of Failure, Control and Value. Moreover, it became clear that we were talking about different states that the same organization could find itself in at different times and in response to different internal and external changes. Further, that the model could be used at a more granular level to describe team level states within an organization.

With apologies to Maslow[1], we created the Harris Hierarchy of Software Development Needs (see Figure 1):

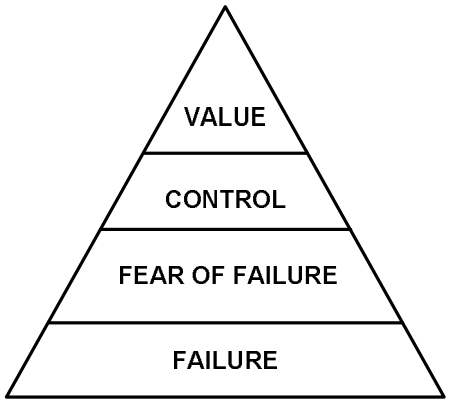


Figure 1: Harris Hierarchy of Software Development Needs

As with Maslowôs Hierarchy, software development teams can move up and down through the tiers as their circumstances change. Broadly, the more mature a software development organization is (Our Tier 4 or ñValueò Tier), the more time it will spend being concerned about delivering value. Immature software development groups (Tier 1 - ñFailureò) lurch from failure to failure dependent on heroes to pull success from the teeth of failure by working ridiculous hours or renegotiating the scope, duration or price of the project[2].

Between these extremes, the Tier 2 (ñFear of Failureò) Software Development (SD) organization experiences frequent, apparently random failures because they do not have repeatable processes - they continually operate in ñFear of Failureò mode in which estimates have ranges of +/- 50% or more. Our Tier 3 ñControlò SD organization is mature enough to have repeatable processes but has not dealt with the governance issues that ensure it continues to deliver value to its internal (and external) clients. You can think of software development maturity in terms of CMMI®, ñPlan, measure, doò or any other framework youõd care to choose.

It is important to recognize that software development organizations in each Tier could need support and coaching but that the nature of their goals is very different!

2.1 Does the Harris Model apply to the whole organization or some subsets?

In general, we suggest that the model can be applied at the level of granularity for which processes and metrics are, more or less, uniformly defined and implemented with vigor. In short, we have found that the model provide a rich source data when applied to ñteamsò within an organization because we have observed that different teams are often at different levels within even quite small organizations.

ñImplemented with vigorò is an important qualifying characteristic of our framework which merits further discussion. Vigor incorporates people into the equation[3]. In our experience good software development requires passion. That passion can be focused on the content, the method, the tools or even the process itself (for metrics ñwonksò like us). We submit that an organization will never successfully achieve or sustain the ñControlò level without some passion for process in the organization. We respectfully suggest that the success of agile development stands on several pillars but one of them is defining a process that developers can be passionate about.

By way of contrast, we considered a few other candidate characteristics such as simply, ñimplementedò or ñimplemented with rigorò but these definitions lacked the active engagement element that we consider to be essential at the higher levels of the model.

Identifying ñteamsò as the expected granularity at which the model can best be applied reinforces the involvement of people in the model since this tends to be the smallest unit of people interaction in a software development project. Interesting, we have often observed situations in which even teams operating at the ñfailureò level can be seen to be ñimplementing with vigor.ò Consider the plate-spinning magician rushing from pole to pole to keep the plates on top from crashing to the ground by adding more spin. No shortage of vigor there! Vigor in this case portends the ability to change.

We have seen several organizations in which the absence of process definition and the dependence on a few heroes was implemented with more vigor that most. It is precisely this combination that allows small organizations such as start-ups or a small group in a major organization to operate at the ñvalueò level by generating significant innovation. However, this combination tends not to be scalable and can make a slightly larger organization (or the same start-up after some success) operate at the ñfailureò level for a period of time.

2.2 So we have the model, how can we use it?

The first intended use of the Harris Hierarchy is as a diagnostic tool to segment software development organization teams into performance levels and to identify the appropriate improvement strategies. To enable this, we have included in this paper some characteristics of teams at different levels, some metrics that might support identification of a level and some causes of upward and downward transitions.

It is important to note at this point that the Harris Hierarchy is really a ñgrey scaleò and that we can certainly see sub-levels within the four major levels. For example, we believe that within the ñFear of Failureò Level, close to the boundary with the ñFailureò level is a sub-level that we call ñFear of Imminent Failure.ò For a company, transitioning from the ñControlò level, we often see that ñFear of Failureò does not capture management attention (it should!) anywhere near as much as ñFear of Imminent Failure.ò Another example of these sub-levels occurs in the ñControlò level ï we have observed that teams operating in the lower half of the ñControlò level close to ñFear of Failureò tend to focus on what we call, ñdefensive control.ò Defensive control focuses on processes, metrics and reporting to avoid being ñcaught outò by management. By contrast, teams operating in the upper half of the ñControlò level, are starting to focus more on processes, metrics and reports that try to measure the value being delivered even if this sometimes makes them look bad in the short term! We believe it is very possible to be at the ñControlò level and not delivering much value!

We believe the Harris Hierarchy has another secondary value for software development leaders in building teams for different types of projects and detecting and interpreting the impact teams moving from one level to another which they will almost certainly do from time to time either upward or downward with without noticing the transition.

2.3 What are some characteristics of teams that are unique to the different levels?

Please see Table 1.

Table 1: Characteristics of teams at different levels of the hierarchy

	Failure	Fear of Failure	Control	Value
Schedule	Partial deliverables of customer schedule	Deliverables close to customer schedule	Deliverables on customer schedule	Deliverables on customer schedule
Content	May be excellent but quantity or quality fall short of customer needs	Inefficiencies of process that are not under control mean that there is not time to deliver the desired quality of content	Exactly what the customer ordered	Exceeds customer expectations
Quality	• Growing defect backlog. • Tier 1 support under constant stress • Severity 1 defects in acceptance testing	• Growing defect backlog. • Tier 1 support under constant stress • Severity 1 defects in acceptance testing	• Stable defect backlog • Predictable defect levels in releases • Predictable turn-around time for bug fixes	• Most defects captured before delivery to customer • Rarely a subject of discussion with the customer
Cost	Estimation[4] is completely unreliable and often relies on SMEôs	Frequent estimating disasters destroy confidence in process	Estimation based on historic data and predictable costs are achieved	Estimation based on historic data and predictable costs are achieved
Customer Satisfaction	Poor	Mixed	Satisfied	Wowed
Team Members	Heroes and villains	ñNervous nelliesò	Efficient and effective professionals	Value Providers
Release Management	Unpredictable	Subject to occasional disasters	Sound	Sound
Risk Management	Non-existent	Too high-level	Structured	Customer-facing

Table 2: Harris Hierarchy Metrics

	Failure	Fear of Failure	Control	Value
Overall	Consistent Performance Below Expectations	Inconsistent Performance	Consistent Performance ALL Metrics	Consistent Improvement of ALL Metrics
Schedule	Inconsistent Performance (generally below baseline)	Dates met and performance generally below baseline	Dates met and performance generally above baseline	Schedule performance consistently above baseline
ROI	Not Measured	Cost Side measured	Cost and Benefit predicted and measured	Full Life Cycle ROI monitored and actions taken to attain
COST Per Unit of Work	Inconsistent Performance (generally below baseline)	Cost performance generally below baseline	Cost performance generally above baseline	Cost performance consistently above baseline
Defects Per Unit of Work	Inconsistent Performance (generally below baseline)	Defect Density performance below baseline	Defect Density performance generally above baseline	Defect Density performance consistently above baseline
Process Assessment	Primarily Ad-Hoc or not formally assessed	Primarily Ad-Hoc but formally Assessed	CMMI (or similar) Level 2 or better	CMMI (or similar) Level 2 or better with improvement plans
Customer Satisfaction	Poor	Inconsistent	Consistent Average or Better	Satisfied Customers
Re-organization Half-life	3 -6 Months	6 ï 12 Months	12 ï 18 Months	18 Months or More

2.4 What Metrics might be indicators of one Harris Hierarchy level or another?

See Table 2. ñPretty goodò metrics provide a spotlight to tell you where you are; ñgoodò metrics provide the view in nuanced manner; ñreally goodò metrics tell you where you are going because you are always changing. In building our model we have found that a robust pallet of metrics is required to meet the definition of ñreally goodò metrics. To work, the pallet must be made up of metrics that change as a team or organization transits through the model and each metric needs to evaluate different aspects of the model than the other metrics in the pallet.

A final characteristic of any metric in the pallet is that it can actually be deployed (not just an academic idea). The metrics pallet supporting the model has two levels. The first level, designated overall, provides an indication of how the metrics should interact in aggregate. For example, the metrics of a team at the Failure level[5] would be consistently below expectations. As the team moves away from Failure they become more inconsistent. As a team moves from Failure to Fear of Failure, performance tends to swing between deliveries that are below expectations, meeting them and occasionally showing flashes of superior performance. Finally, arriving at the Value level, all metrics should be above expectations and still show consistent improvement.

Table 2 names each metric and describes how the performance varies at each level. The authors recommend creating a baseline for each metric. This will allow you to validate your appraisal of a team level and to determine whether changes to the team are effective.

2.5 What might a Harris Hierarchy Assessment look like?

We can anticipate a number of different ways of looking at the data from a team-by-team evaluation of Harris Hierarchy levels but, as a starting point, we always ask ourselves, ñWhat decision will be made when this information is presented to management?ò In this case, when we report to management that we see their teams operating at a variety of Harris Hierarchy levels, they tend to ask something like, ñDo I need to do anything about it today?ò and then ñHow can I prioritize?ò Based on this type of reaction, our favorite initial chart is a bubble chart that captures the teamsô Harris Hierarchy levels, the business impact of the project they are working on and the size of the team in terms of full time equivalent people (see Figure 2).

An analysis of this this presentation clearly shows that teams B and C should be the first candidates for attention because of the potential to impact the business. Teams D & E need attention but it might be more efficient to disband them than to try to improve then given the relatively low business impact. Team A is interesting and a type of team quite often. The team is a reasonable size and it is operating at a high level in the hierarchy therefore it is usually the type of team that gets highlighted as evidence that ñwe are doing just fine ï thanks.ò Thatôs certainly true for team A (they may have done CMMI, ISO 9001 or the like). However, especially given where it sits in terms of potential business impact, is that really true for the organization as a whole? A discussion of redirection would make sense for teams in this quadrant.

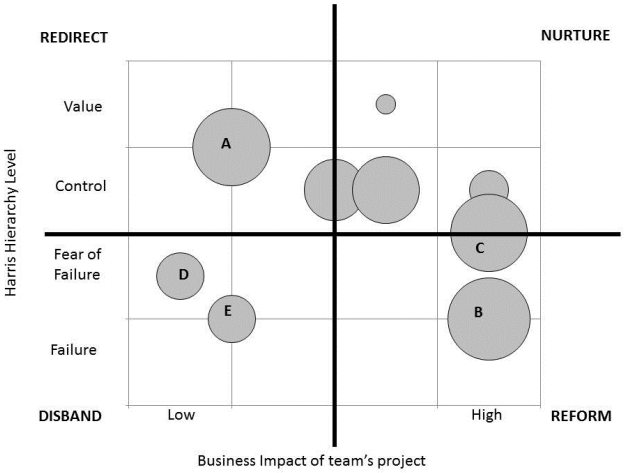


Figure 2: Example Harris Hierarchy Assessment

2.6 What evidence do we have of the applicability of the model so far?

2.6.1 Provider of large, mission-critical software products to the telecoms industry

When we were introduced to this company, they had some of their teams operating at the ñControlò level (see Figure 3 ï teams A, B , C & D) while some were operating at the ñFailureò Level (teams E & F). Unfortunately, teams E and F were very visible to customers because they were new products with heavy user interface elements. The ñFailureò teams were part of acquisitions for which the software development management team did not feel consulted and, hence, they did not regard themselves as accountable. Clearly, the solution here was to apply the ñControlò practices to the new teams. However, there was resistance at first because of the fear that the new teams would ñmess upò the nice metrics of the old teams. One of the ways that the ñControlò level had been achieved was to virtually exclude the rest of the company from the development process. As a new organizational structure with new personalities was introduced, control over the content and budgets for software development moved outside of the software development organization and the resulting culture change dropped one team (team D) from ñControlò level down to ñFear of Failureò level. We have seen this happen in other

organizations where the ñControlò level is sustained through a rigid, almost bureaucratic, approach which does not deal with change well.

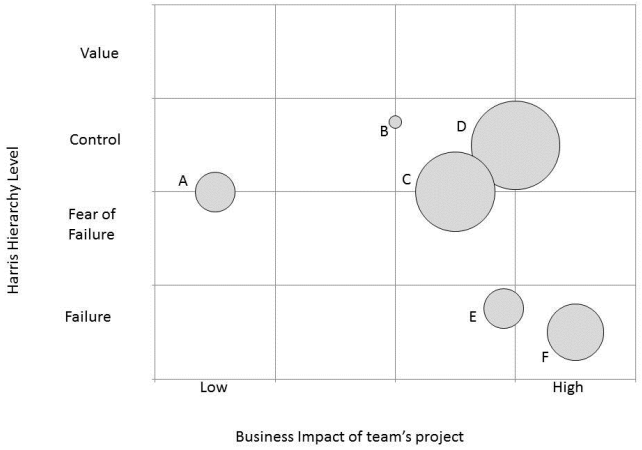


Figure 3: Real Harris Hierarchy Assessment

2.6.2 Provider of customized software solutions to the Department of Defense using SCRUM

In the eyes of their clients, this company clearly operated at the ñValueò level which won them more business. However, as the size of their projects grew due to the absence of controls, they quickly dropped to the ñFear of Failureò level. One of the areas that exhibited the impact of the reduction of capability was release management. Without controls they quickly lost control of release management which was an essential part of their Agile SCRUM value proposition. Implementing a lean version of parts of CMMI enabled them to return to the ñControlò level and from there to plan a return to the ñValueò level.

2.6.3 Provider of large, mission-critical software products to the banking industry

Prior to implementing a CMMI program to provide a focus on structure and discipline, this company operated at the ñFailureò level even though they were perceived as highly successful. The issue was that only one of forty clients would provide a reference and the relatively small group of employees who could successfully execute was run ragged as the company grew. After implementing the CMMI program, the company moved up to the ñControlò level but soon dropped back to the ñFear of Failureò level as management focus on the CMMI program shifted elsewhere.

2.7 What keeps companies in one state rather than transitioning?

Our early sharing of this model outside of the research group most often generates a question of how stable is the ñfailureò level. Our observations suggest, it is an unstable level. Organizations or teams must eventually move up or collapse. The rate of collapse and rebuilding can be measured in the average half-life of organizational restructuring or CIO turnover. The lower the half-life (use the last three reorganizations as a baseline) the larger the impact of failure caused by churn on the organization. Unfortunately, we see all too often that an excessive presence of passion in the form of heroes or an excessive absence of passion for the level of discipline that can stop the pain. In these cases organizational inertia can cause teams to wallow in the failure level far longer than we might like or expect. Either failure is ñunderstandableò because the team tried so hard or ñnormalò because itôs always been that way, hasnôt it?

At the higher levels, companies stay in one state by keeping their discipline alive and *relevant* to the day to day work of the development teams. Here again, organizational stability is a quick means of measuring the impact of higher levels.

2.7.1 What can cause upward transitions?

- Intervention
  - Positive reinforcement of best practice
  - Change of non-productive behavior to best practice
  - Near death experiences
- Success
  - Innovative or accidental behavior that works (and is identified and reinforced)
- Removal of control from the team
  - Breaking of ñtraditionalò behavior patterns for example moving away from a subject matter expert generated estimation model to a centralized parameter-based, historic data-driven estimation model.[6]
  - Implementing self-directed team models based on shared goals[7]
- Investment
  - In the implementation of specific best practices such as CMMI or Agile. While there is always an internal component to this investment, a key to success here is to bring in the right external consultants as software development experts and change agents.
- Outsourcing[8]
  - Yes, itôs true! Sometimes the outsourcing of software development to a third party allows the organization to leverage the best practices of ñexpertsò whose sole business is software development. Sometimes even the need to define processes to make outsourcing possible improves what had gone before.

2.7.2 What can cause downward transitions?

- Intervention

- A sense of lack of control caused by the intervention of ñmanagementò can de-motivate the most passionate contributors
- Lack of consistency of purpose (intervention of the day)

- Success
  - Different size teams require different processes to generate the same results. This cannot be ignored as success causes the team to grow.
  - Past success based on ad hoc processes or heroism in a dynamic environment. Past performance does not ensure future success in this environment.
- Removal of control from the team
  - Successful software development requires detailed knowledge more than most engineering activities and the most detailed knowledge is usually at the bottom of a hierarchical team not at the top. Mangers ignore this reality at their peril.
- Unrealistic expectations[9]
  - Unrealistic expectations are a guarantee of failure and repeated failure is a trigger for a downward transition.
- Outsourcing
  - Outsourcing bad software development practices will not improve them
  - Introducing outsourcing necessarily involves changes to process flows and the threat of mistakes

3 Conclusions

The Harris Hierarchy of Software Development Needs is a new concept that we believe adds value and should be included in organizational tool sets to help managers to identify software development teams with issues through the prism of a disciplined framework.

The ideas that we have provided in this paper are intended as starting points for further development by practicing software development managers.

4 References

[1] Maslowôs Hierarchy of Needs: http://en.wikipedia.org/wiki/Maslow's_hierarchy_of_needs

[2] Frederick Manzer, ñThe Impact of Fear on Project Success,ò Ask Magazine, No.24: 42

[3] Kim S. Cameron, Robert E. Quinn, Diagnosing and changing organizational culture: based on the competing values framework, John Wiley and Sons, 2006, P11

[4] Frank McGarry, Steve Burke, Bill Decker, Measuring the Impacts Individual Process Maturity Attributes Have on Software Products, 5th International Symposium on Software Metrics

[5] Frederick Manzer, ñThe Impact of Fear on Project Success,ò Ask Magazine, No.24: 43

[6] Paulk, Curtis, Chrissis, Weber, Capability Maturity Model℠ for Software, Version 1.1, CMU/SEI-93-TR-024,15

[7] Pun Santa, The impact of cross-functional teams on operational performance after the implementation of intelligent information systems, IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems, September 2009

[8] http://www.indiapractice.com/offshore_outsourcing_risk_factors Cross-Cultural Risk Factors in Offshore Outsourcing

[9] Watts S Humphrey, Five Reasons Why Software Projects Fail. Computerworld, May 20, 2002, http://www.computerworld.com/s/article/71209/Why_Projects_Fail

(Dis)economies of Scale in Business Software Systems Development and Enhancement Projects

Beata Czarnacka-Chrobot
Department of Business Informatics, Warsaw School of Economics, Warsaw, Poland, bczarn@sgh.waw.pl

Abstract - *In the software engineering literature it is commonly believed that economies of scale do not occur in case of software Development and Enhancement Projects (D&EP). Their per-unit cost does not decrease but increases with the growth of such projects product size. Thus this is diseconomies of scale that occur in them. The significance of this phenomenon results from the fact that it is commonly considered to be one of the fundamental objective causes of their low effectiveness. This is of particular significance with regard to Business Software Systems (BSS) D&EP characterised by exceptionally low effectiveness comparing to other software D&EP. Thus the paper aims at answering the following two questions: (1) do economies of scale really not occur in BSS D&EP? (2) If economies of scale may occur in BSS D&EP, what factors are then promoting them? These issues classify into economics problems of Software Engineering Research and Practice (SERP).*

Keywords: (dis)economies of scale, business software systems development and enhancement projects, software size metrics, functional size measurement, economies of scale factors

1 Introduction

In the literature on software engineering it is commonly believed that software Development and Enhancement Projects (D&EP) are characterised by the absence of economies of scale – contrary to the majority of other business undertakings, including e.g., building engineering projects, where the per-unit cost decreases while productivity increases with the increase of product size, at least to a certain limits of size. Meanwhile in case of software D&EP, per-unit cost does not decrease yet increases (productivity decreases) with the growth of product size, therefore this is diseconomies of scale that occur in them. This applying even to the smallest product sizes: "Per-unit cost of larger software system is more expensive than analogous cost of a smaller system" [1].

This phenomenon, considered as immanent to software D&EP, appears to be of significance as much as it is regarded as one of the fundamental objective, i.e., resulting from the specificity of such projects as compared with other engineering projects, causes of their low effectiveness. It is of particular importance in case of software D&EP whose products are Business Software Systems (BSS), as their effectiveness is exceptionally low comparing to D&EP delivering other types of software products.

As indicated by the results of the Standish Group analyses success rate for application D&EP has never gone beyond 35% [2]. It means that majority of them either end up with total failure, or they exceed costs and/or time estimated as well as they lack critical functions/features. The Standish Group estimates that now only 32% of application D&EP worldwide turn out successful while products delivered as a result of nearly 45% of them lack on average 32% of the required functions and features, the planned time of product delivery is exceeded by nearly 80% on average and the estimated budget - by approx. 55% on average. On the other hand, analyses by T.C. Jones plainly indicate that those software D&EP, which are aimed at delivering BSS, have the lowest chance to succeed [3]. The Panorama Consulting Group, when investigating in their 2008 study the effectiveness of ERP (Enterprise Resource Planning) systems projects being accomplished worldwide, revealed that 93% of them were completed after the scheduled time while as many as 68% among them were considerably delayed comparing to the expected completion time [4]. Merely 7% of the surveyed ERP projects were accomplished as planned. Comparison of actual versus planned expenses has revealed that as many as 65% of such projects overran the planned budget. Only 13% of the respondents expressed high satisfaction with the functionality implemented in final product.

Meanwhile BSS are not only one of the fundamental application areas of software engineering, also their development/enhancement often constitutes serious investment undertaking: spending on BSS D&EP may considerably exceed the expense of building offices occupied by companies commissioning such systems, and in extreme cases, even 50-storey skyscraper, roofed football stadium, or cruising ship with a displacement of 70.000 tons [5]. Exceptionally low effectiveness of BSS D&EP as compared to other types of software projects, with their costs being considered, leads to the substantial financial losses, on a worldwide scale estimated to be hundreds of billions of dollars yearly.

Thus the paper aims at finding answer to the following two questions:

1. Is it true that economies of scale do not occur in BSS D&EP, as it is commonly assumed in the software engineering literature, and therefore can this specific phenomenon be treated as objective cause justifying the low effectiveness in case of such projects?
2. If economies of scale may occur in BSS D&EP, what factors are then promoting them?

2 Business software systems size metrics

Whenever investigating into (dis)economies of scale, it is important to weigh the costs of product development/enhancement against the product size expressed in appropriate size units. In economics or other (than software engineering) engineering disciplines the product size unit as a rule is evident whereas answer to the question about product size units in case of software products is not evident at all, and these units are of significance to the results of such investigations. The question then arises: in what units the size of software product, including BSS, is measured?

Basic approaches to the size measurement of every software product may be reduced to perceiving it from the perspective of (see also [6]):

- Length of programmes, measured by the number of the so-called programming (volume) units. These units most of all include source lines of code. However, these units measure neither size of the software products nor their complexity but only the attribute of "programme length" yet thus far these are them that in practice have been employed most often with regard to the software size [7].
- Software product construction complexity, measured in the so-called construction complexity units. Most of hundreds of such metrics having been proposed are limited to the programme code yet currently these units are used mainly in the form of object points [7]. These points are assigned to the construction elements of software (screens, reports, software modules) depending on the level of their complexity.
- Functionality of software product, expressed in the so-called functionality units. They most of all include Function Points (FP), which are assigned to the functional elements of software (functions and data needed to complete them) depending on the level of their complexity. Moreover, the measure of software functionality is *not* the so-called *adjusted* function points, present in some of the methods dedicated to its measurement, which are designed to correct functional user requirements with non-functional requirements.

Synthetic comparison of various software size metrics against a background of key requirements set for them was presented in Table 1.

The right metric of software product size has been sought out for several decades now. Many years' verification of various approaches' reliability and objectivity showed that what for now deserves standardization is just the concept of software size measurement based on its functionality – being an attribute of first priority to the client. Due to the empirically confirmed effectiveness of such approach, it was in the last years normalized by the ISO (International Organization for Standardization) and IEC (International Electrotechnical Commission), and turned into the six-part international standard ISO/IEC 14143 for the so-called software Functional Size Measurement (FSM) [8].

Details displayed in Table 1 clearly indicate the reasons why functionality units were recognised as the most appropriate metric of software product size not only by the ISO/IEC but also, among others, by Gartner Group [9] as well as by International Software Benchmarking Standards Group (ISBSG) [10]. They show no limits being characteristic of programming units and construction complexity units – although one may have reservations as to their versatility and relatively high complexity of the methods based on them. However, it is hard to expect that the method of measurement of software products, by nature being complicated, would be effective yet simple.

Table 1. Synthetic comparison of software size metrics

Requirement towards metrics	Programming units	Construction complexity units	Functionality units
Unequivocalness of definition	Freedom in formulating definitions (differences as big as even 5:1)	Depending on the method	In methods normalized by ISO/IEC
Possibility to make reliable prognosis on the size relatively early in the life cycle	Possibility to calculate programme length only for the existing code	None – with regard to programming units and object points	As early as at the stage of requirements specification
Base for the reliable evaluation of the all phases work effort	Final programme length does not fully reflect the whole work done	Final software size does not fully reflect the whole work done	Relatively high reliability as early as at the stage of requirements specification
Software size being independent of the technology employed	Programme length determined by the language employed	Size being dependent on the technology employed	Size depends on functional user requirements
Measuring size in units being of significance to a client	No significance to a client	Secondary significance to a client	Measurement from the point of view of a client
Possibility to measure all software categories	Yes	Depending on the method	Depending on the method
Easiness of use	Yes	No	No

Source: Author's own analysis.

The set of rules for software FSM enclosed in the ISO/IEC 14143 norm provides key definitions, characteristics and requirements for FSM, and defines Functional Size Measurement Method (FSMM) as a specific FSM implementation defined by a set of rules, which conforms to the mandatory features of such measurement. The first part of this standard defines also indispensable principles upon which the FSMM should be based – fundamental one is the definition of functional size, which is understood as "a size of the software derived by quantifying the functional user requirements", while "the Functional User Requirements (FUR) represent the user practices and procedures, that the software must perform to fulfil the user's needs. FUR exclude Quality Requirements and any Technical Requirements" [8].

There are about 25 variants of the FSM techniques having been developed, however only five of them have been now acknowledged by the ISO/IEC as conforming to the rules laid down in the ISO/IEC 14143 norm and certified as international standard, namely: (1)

International Function Point Users Group (IFPUG) method approved in the ISO/IEC 20926 standard [11]; (2) Mark II (MkII) function point method proposed by the United Kingdom Software Metrics Association (UKSMA) and normalized in the ISO/IEC 20968 standard [12]; (3) Netherlands Software Metrics Association (NESMA) function point method approved in the ISO/IEC 24570 standard [13]; (4) Common Software Measurement International Consortium (COSMIC) method certified in the ISO/IEC 19761 standard [14], now being revised; and (5) FSM method developed by the Finnish Software Metrics Association (FiSMA) and normalized in the ISO/IEC 29881 standard [15].

The first three methods listed above are accepted by the ISO/IEC not in full versions, as proposed by the organizations developing them, but in part, however in the most important part with respect to the software functional size measurement [8, Part 6] – that is why they are called the first-generation FSMM. In the approaches proposed by IFPUG, UKSMA and NESMA these methods involve also delineating of the so-called value adjustment factor, which is supposed to adjust functional size, being measured with the use of *Unadjusted* Function Points (UFP), to the environment of specified project by taking technical and quality requirements into consideration – thus final result is presented in the so-called *Adjusted* Function Points (AFP) [16, Part 5]. Yet this part of these methods has not been approved by the ISO and IEC – as these organizations' assumptions exclude the fact of FSM depending on requirements of this type. On the other hand, the COSMIC and FiSMA methods were recognized as international standard entirely as the phase of adjustment of functional size to non-functional requirements does not exist in them, which means that function points of the COSMIC method and FiSMA method always depict the functional size of software product ([8, Part 6][15]). That is why these two methods are called the second-generation FSMM.

The FSMM standardized by the ISO/IEC differ in terms of software measurement capabilities with regard to different software classes (functional domains), but all of them are adequate for BSS. Also, these methods provide rational basis for methodologies supporting BSS D&EP scope management (for more details see [17]).

Summing up, the only metric of software product size being so far recognised as sufficiently objective and reliable are function points, which allow to measure product functional size by using one of the five FSM methods accepted by ISO/IEC – in case of first-generation methods this is *unadjusted* FP being used for that purpose.

3 Studies proving diseconomies of scale in software D&EP

Among exemplary studies indicating diseconomies of scale in software D&EP are those of the following authors: H.D. Knöll and J. Busse [18], L.H. Putnam and W. Meyers [19], M. Cusumano et al. [20], S. McConnell [21], T.C. Jones [22], and B. Boehm et al. [23]. All these analyses prove the occurrence of diseconomies of scale in software D&EP:

- with regard to software product size expressed in programming units ([19][20][21][23]) or in *adjusted* function points ([18][22])
- already at the smallest sizes of software product.

Based on data contained in [18] one may create graph presented in Fig.1. It plainly indicates that in the case of exemplary categories of software systems the project effort grows exponentially with the increase of software product size expressed in IFPUG adjusted FP. On the basis of data for banking software system (being BSS), presented in Fig. 1, for which the fastest increase of work effort may be noticed, one may, on the other hand, derive dependencies pictured in Fig. 2 and Fig. 3. As it may be seen, with the increase in the product size of such system being expressed in IFPUG adjusted FP, per-unit effort goes up while productivity decreases, as early as with the smallest values of product size.

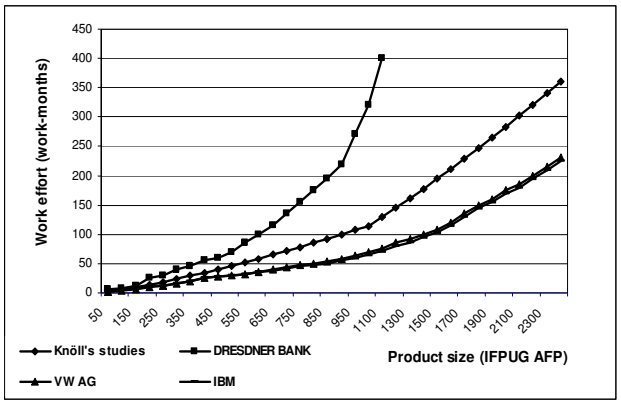


Fig. 1. Dependency between software D&EP work effort and product size expressed in IFPUG adjusted FP for exemplary categories
Source: Author's own analysis based on [18].

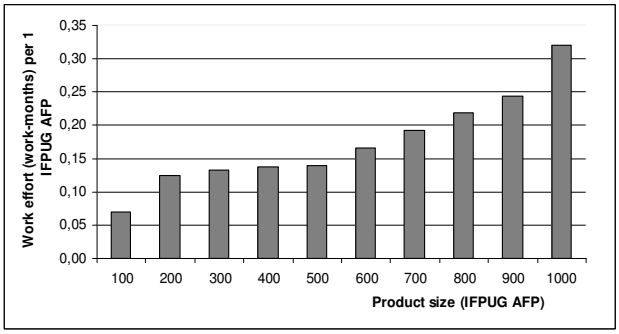


Fig. 2. BSS D&EP per-unit work effort versus product size expressed in IFPUG adjusted FP for banking software system
Source: Author's own analysis based on [18].

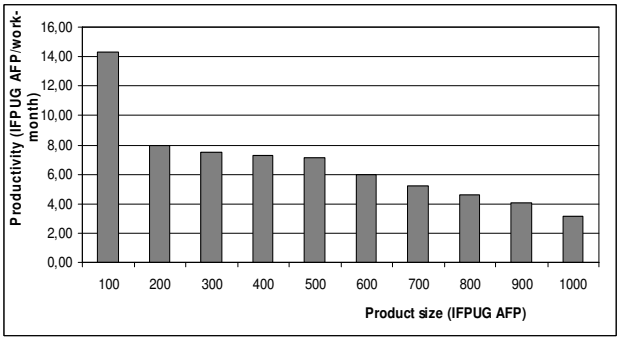


Fig. 3. BSS D&EP productivity versus product size expressed in IFPUG adjusted FP for banking software system
Source: Author's own analysis based on [18].

The fact of the described dependencies being shaped in similar way was indicated also by the remaining studies mentioned above (as well as by many other studies),

including those where programming units were chosen to express software product size.

What is considered to be the main reason for diseconomies of scale to occur in software D&EP is regularity according to which the number of communication channels between project participants (n) increases with the growth of the software system being constructed so that the amount of work increases proportionally to [n x (n-1)]/2 (actual number of communication channels) [21]. Other significant factors of diseconomies of scale (so-called scale factors), whose influence depends on software product size (the larger the product, the stronger their influence), include, according to McConnell, the following: immaturity of software processes, inappropriate risk management, project precedentness, poor collaboration between project participants, and inflexible user requirements.

4 Studies proving economies of scale in BSS D&EP

In 2009, the International Software Benchmarking Standards Group in cooperation with COSMIC published results of the studies, which in case of BSS D&EP contradict the commonly adopted thesis on the absence of economies of scale in software D&EP [24]. These conclusions apply to BSS being measured in COSMIC function points.

The ISBSG is a non-profit organization that was established in the second half of the 90s of the last century with the mission to improve processes of software projects execution in business companies and public administration institutions as well as to support development of software organizations [25]. This mission is being fulfilled by developing, maintaining and exploiting three kinds of repositories with benchmarking data regarding software product and process measurement. One of them, the largest one, comprises data for software D&EP, the second one – data for software maintenance and support applications, while the third one – data for software package acquisition and implementation. These repositories, standardized according to the ISO/IEC 15939 norm [26], verified and representative of current technology, are meant to support decision-making process in the area of software engineering. Data in the repository for software D&EP have been obtained from over 5000 projects completed in more than 25 countries. Generally, they concern projects delivering business applications as their products. For all projects the software product size is expressed as functional size of one of the methods recognized by ISO/IEC.

Studies on (dis)economies of scale in BSS D&EP were based on the verified data from nearly 300 of such projects, of which more than half are projects being completed in the banking sector while the remaining ones took place in the sector of public administration, insurance, engineering, health care, and trade. Approx. 50% of the considered projects are BSS new development projects with product size ranging from 10 to 1670 COSMIC FP, with work effort ranging from 200 work-hours to 32 work-years (median equals 3 work-years) and duration of 1 to 65 months (median equals 9 months). The rest are BSS enhancement projects with product size ranging from 3 to 750 COSMIC FP, with work effort ranging from 24 work-hours to over 15 work-years. The results coming from the ISBSG and COSMIC studies are presented in Fig. 4 and Fig. 5.

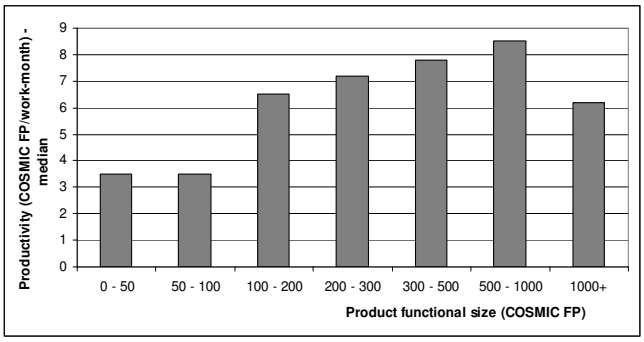


Fig. 4. Productivity versus software product functional size expressed in COSMIC FP for BSS new development projects
Source: [24].

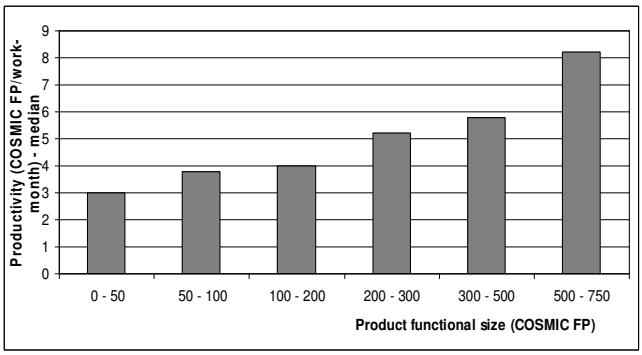


Fig. 5. Productivity versus software product functional size expressed in COSMIC FP for BSS enhancement projects
Source: [24].

Charts presented in Fig. 4 and Fig. 5 indicate that *economies of scale occur both in BSS new development projects as well as in BSS enhancement projects*:

- In BSS new development projects up to the size of 1000 COSMIC FP, then the productivity median decreases (however for product size over 1000 COSMIC FP there were data available for a little number of projects only).
- In BSS enhancement projects at least up to the size of 750 COSMIC FP (data were not available for larger sizes of such projects' products).

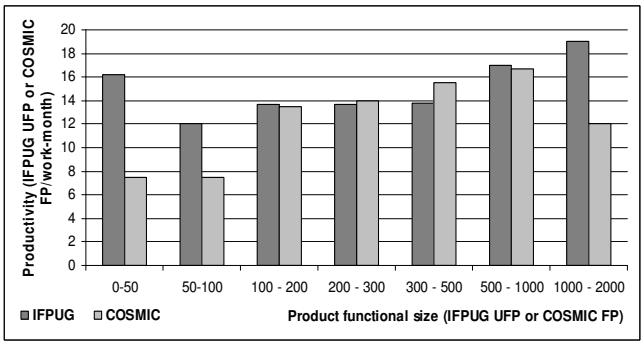


Fig. 6. Comparison of dependencies between productivity and software product functional size expressed in IFPUG unadjusted FP and COSMIC FP (BSS new development projects)
Source: [24].

What's also interesting is the comparison of dependencies between productivity and product functional size expressed in IFPUG unadjusted FP and COSMIC FP, which may be seen in Fig. 6. As it indicates, in case of

both FSMM the productivity increases with the growth of product functional size: in the range of 0 to 1000 COSMIC FP and in the range of 50 to 2000 IFPUG UFP.

5 Factors promoting economies of scale in BSS D&EP

According to the author of this paper, among fundamental factors of economies of scale in BSS D&EP should be first of all mentioned:

1. Maturity of organization and BSS D&EP processes
2. Use of sufficiently objective and reliable method of BSS size measurement
3. Undertaking small BSS D&EP.

5.1 Marurity of organization and BSS D&EP processes

Criteria for data collection in ISBSG take into account only organizations employing the methods of software FSM. These organizations are considered more mature than the others within the meaning of CMMI (Capability Maturity Model Integration [27]) – as they comprise programmes concerning implementation of software measures. The latest version of the CMMI for Development of 2006 is strongly oriented towards measurement of software processes and products. It regards measurement as being fundamental to achieving the next levels of organization maturity: the higher the level, the stronger its orientation towards quantitative approach. Measurement and analysis process area, distinguished explicitly, is ascribed to the second level of organization maturity (managed).

However H. von Loon estimates that not many software organizations achieve high maturity levels, this being a problem applying to Europe in particular (in the ISBSG repository this is data from the USA, Japan and Australia that dominate [28]). Analyses presented in [29] prove that what for the companies at initial level of maturity appears to be one of the fundamental causes of difficulties in achieving maturity level 2 is the lack of software measurement procedures, with the metric of software product size being the one skipped the most often.

Meanwhile results of the studies concerning effectiveness of CMMI, carried out not only by the Software Engineering Institute [30], being this model's author, but also by D. Rico [31], reveal that in organizations at the high level of maturity:

- Costs and time of project completion are estimated more accurately thanks to the appropriate collection of reliable benchmarking data: organizations at maturity level 5 (optimizing) practically do not exceed estimates whereas organizations at maturity level 1 (initial) exceed the estimated time by 150% on average, and the estimated cost by nearly 200% on average.
- Quality of final products increases due to lower number of errors and this being thanks to the control of the intermediate products quality initiated at the earliest phases of project lifecycle.
- Cost of improving bad quality of intermediate products (maintenance cost) decreases due to the lower number of errors: average cost of error correction in organizations at maturity level 5 amounts to approx. 4% of the total software development costs while in organizations at initial level it amounts to over 50% of such costs.
- Total cost of software development decreases as a result of the decrease in the cost of improving bad quality of products: average cost of developing one function is more than 3 times lower in organizations at the highest level comparing to that in organizations at the lowest level.
- Productivity grows as a result of the decrease in the above mentioned per-unit cost.
- Total time of products completion gets reduced due to lower number of errors and therefore reduction of the time designed for correcting them.

5.2 Use of sufficiently objective and reliable method of BSS size measurement

If BSS D&EP product size is not appropriately measured then the results for per-unit cost/per-unit work effort/productivity of such projects will be incorrect. Analysis presented above indicates that the only methods of software product size measurement recognised as sufficiently objective and reliable are FSMM having been normalised by the ISO/IEC. They have to match with the proper functional domain (see Table 2 – as indicated in there, all standardized FSMM are adequate for BSS) and be applied in accordance with the appropriate norms.

Table 2. The ISO/IEC standards for FSMM versus functional domains

FSMM	Functional domains specified in the norm	Constraints indicated in the norm
ISO/IEC 20926 - IFPUG method (unadjusted FP)	All software classes	None
ISO/IEC 20968 - UKSMA method	For any type of software provided that the so-called logical transactions may be identified in it (the rules were developed as intended for BSS).	The rules support neither complex algorithms characteristic of scientific and engineering software nor the real-time systems.
ISO/IEC 24570 – NESMA method	All software classes	None
ISO/IEC 19761 - COSMIC-FFP method	- Data-driven systems, e.g., business applications for banking, insurance, accounting - Real-time systems (time-driven systems), e.g., telephone exchange systems, embedded software, operation systems - Hybrid solutions combining both of the above, e.g., real-time systems of airline tickets booking.	- Systems with complex mathematical algorithms or with other specialised and complex rules (e.g., expert, simulation, self-learning systems) - Systems processing continuous variables (audio, video) For above-mentioned domains it is possible to modify the method so that it may be used locally.
ISO/IEC 29881 - FiSMA method	All software classes	None

Source: Author's own analysis based on [8, Part 6].

What also should be taken into account is that 1 IFPUG UFP does not equal 1 COSMIC FP. Dependencies between these units are only approximate and differ for different types of projects as well as for different kinds and different sizes of product. So there is no possibility of exact conversion of the results of both methods using

mathematical formula. One of the approaches towards conversion is conversion based on statistical formula (see e.g., [32]), but this issue requires further investigations.

5.3 Undertaking small BSS D&EP

Figure 6 indicates that the size of BSS (its fragment) being developed one-off should not exceed certain limit: for BSS new development projects being 1000 COSMIC FP or 2000 IFPUG UFP on average (exemplary estimates for functional sizes of many software products may be found in [33]). Such approach requires appropriate estimation of product size threshold point, with type of project and software product as well as iterative approach to development/enhancement being taken into account.

Small software products favour early involvement of a little users group in the project activities, lower the dynamics of requirements changes, facilitate precise defining of real, clear goals and expectations of a client as well as responsibility for the product, are being accomplished by small development teams which results in lower effort of interactions between project participants (see section 3) and therefore they are easier to manage.

There are numerous studies proving higher effectiveness of small BSS D&EP. For instance, according to T.C. Jones, the fact that BSS D&EP are characterised by significantly lower effectiveness comparing to D&EP delivering other types of software products, for the most part results from the large size of such projects while small software products D&EP decidedly more often lead to successful end [34]. Moreover, what Standish Group considered to be the major cause of the increase in the effectiveness of application D&EP execution during 1994-2008 by 100% (from the level of 16% to 32%) is minimisation of their size [35]. Eighty per cent of successful projects prove having work costs lower than USD 3 million while projects with work costs below USD 750.000 have 71% of the chance to succeed whereas for projects with work costs ranging from USD 750.000 to USD 3 million the chance is 38% and for those costing more than USD 10 million – 0% [2]. This is a consequence of using iterative approach to development, agile approach in particular, on a more frequent basis over time. Thus particular attention should be given to the prioritisation of functions and selection of those being of significance to the fulfilment of project's goal – since on average only about 20% of functions and features specified are used often and 50% of them are hardly ever or never used at all [36].

6 Concluding remarks

Answering, based on the analysis carried out in the paper, the two questions posed in the introduction it should be stated that:

1. *Economies of scale can occur in BSS D&EP* – contrary to the view being common in the software engineering literature, which *in case of BSS D&EP questions the importance of diseconomies of scale factor as an objective cause justifying their low effectiveness*.
2. Fundamental factors promoting economies of scale in BSS D&EP include:
   - Maturity of organization and BSS D&EP processes
   - Use of sufficiently objective and reliable method of BSS size measurement
   - Undertaking small BSS D&EP.

In addition to the above, it should be mentioned that in the ISBSG and COSMIC studies that were quoted no economies of scale were revealed for real-time and component software D&EP [24].

It also should be pointed out that when analyzing the ISBSG repository with benchmarking data for software D&EP one should keep in mind that data gathered by the ISBSG are representative not for all of such projects but rather tend reflect "better than average" projects, which results from the following facts:

- Criteria for gathering data in the ISBSG repository refer only to these organizations, which are employing FSMM while apparently these organizations are more mature than others as they carry out programmes concerning implementation of software metrics.
- Data to be included to the ISBSG repository are chosen by the organizations themselves – they may choose projects that are typical of them as well as projects characterised by the best attributes.
- The ISBSG repository does not include a good deal of data about really large software D&EP.

Further works should be oriented first of all towards verification of conclusions coming from the analyses of ISBSG and COSMIC – concerning not only BSS D&EP, but also other types of software projects, the analysis of the impact of particular factors on economies of scale in BSS D&EP as well as continuous search for possibilities of BSS D&EP effectiveness growth.

7 References

[1] Z. Szyjewski. "Metodyki zarządzania projektami informatycznymi" ["Methodologies of IT Projects Management"], Placet, Warsaw, 2004.

[2] Standish Group. "CHAOS Summary 2009", West Yarmouth, Mass., 2009.

[3] T.C. Jones. "Patterns of Software Systems Failure and Success", International Thompson Computer Press, Boston, MA, 1995.

[4] PCG. "2008 ERP Report, Topline Results", Panorama Consulting Group, Denver, 2008.

[5] T. C. Jones. "Software project management in the twenty-first century", Software Productivity Research, Burlington, 1999.

[6] B. Czarnacka-Chrobot. "The Economic Importance of Business Software Systems Size Measurement"; in: Proceedings of the 5th International Multi-Conference on Computing in the Global Information Technology (ICCGI 2010), M. Garcia, J-D. Mathias, Eds., pp. 293-299, IEEE Computer Society, Los Alamitos-Washington-Tokyo, 2010.

[7] M.A. Parthasarathy. "Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects", Addison Wesley, 2007.

[8] ISO/IEC 14143 Information Technology – Software measurement – Functional size measurement – Parts 1–6, ISO, Geneva, 1998-2007.

[9] Gartner Group. "Function Points Can Help Measure Application Size", Research Notes SPA-18-0878, 2002.

[10] International Software Benchmarking Standards Group. "The ISBSG Report: Software Project Estimates – How accurate are they?", ISBSG, Hawthorn, VIC, 2005.

[11] ISO/IEC 20926 Software and systems engineering - Software measurement - IFPUG functional size measurement method 2009, ISO, Geneva, 2009.

[12] ISO/IEC 20968 Software engineering – Mk II Function Point Analysis – Counting practices manual, ISO, Geneva, 2002.

[13] ISO/IEC 24570 Software engineering – NESMA functional size measurement method version 2.1 – Definitions and counting guidelines for the application of Function Point Analysis, ISO, Geneva, 2005.

[14] ISO/IEC 19761 Software engineering – COSMIC-FFP – A functional size measurement method, ISO, Geneva, 2003.

[15] ISO/IEC 29881 Information Technology – Software and systems engineering – FiSMA 1.1 functional size measurement method, ISO, Geneva, 2008.

[16] International Function Point Users Group. "Function Point Counting Practices Manual, Release 4.3", Part 0-5, IFPUG, NJ, January 2010.

[17] B. Czarnacka-Chrobot. "Methodologies Supporting the Management of Business Software Systems Development and Enhancement Projects Functional Scope"; in: Proceedings of the 9th International Conference on Software Engineering Research and Practice (SERP'10), H.R. Arabnia, H. Reza, L. Deligiannidis, Eds., pp. 566–572, CSREA Press, Las Vegas, Nevada, USA, 2010.

[18] H.D. Knöll, J. Busse. "Aufwandsschätzung von Software-Projekten in der Praxis", Mannheim 1991.

[19] L. Putnam, W. Meyers. "Industrial Strength Software: Effective Management Using Measurement", IEEE Computer Society Press, Washington, DC, 1997.

[20] M. Cusumano et al. "Software Development Worldwide: The State of the Practice", IEEE Software, November/December 2003, pp. 28-34.

[21] S. McConnell. "Software Estimation: Demystifying the Black Art", Microsoft Press, 2006.

[22] T.C. Jones. "Estimating Software Cost", 2 edition, McGraw-Hill Osborne Media, 2007.

[23] B. Boehm et al. "Software Cost Estimation with Cocomo II", Prentice Hall, 2009.

[24] Ch. Symons. "The Performance of Real-Time, Business Application and Component Software Projects", COSMIC and ISBSG, September 2009.

[25] P.R. Hill. "Some Practical Uses of the ISBSG History Data to Improve Project Management", ISBSG, Hawthorn, VIC, 2007.

[26] ISO/IEC 15939 Systems and software engineering – Measurement process, ISO, Geneva, 2007.

[27] CMMI Product Team. "CMMI for Development, version 1.2", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 2006.

[28] International Software Benchmarking Standards Group. "Data Demographics Release 11", ISBSG, Hawthorn, Australia, June 2009.

[29] C.A. Dekkers, E. Emmons. "How Function Points Support the Capability Maturity Model Integration"; in: CrossTalk. The Journal of Defence Software Engineering, February 2002, pp. 21–24.

[30] D.L. Gibson, D.R. Goldenson, K. Kost. "Performance Results of CMMI-Based Process Improvement", Software Engineering Institute, Carnegie Mellon University, Pittsburgh, August 2006.

[31] D.F. Rico. "ROI of software process improvement: metrics for Project Managers and Software Engineers", J. Ross Publishing, February 2004.

[32] Common Software Measurement International Consortium. "The COSMIC Functional Size Measurement Method, version 3.0, Advanced and Related Topics", COSMIC, Québec, December 2007.

[33] T.C. Jones. "A New Business Model for Function Point Metrics, Version 10.0", August 16, 2009.

[34] T.C. Jones. "Software Assessments, Benchmarks, and Best Practices", Information Technology Series, Addison-Wesley, 2000.

[35] Standish Group. "CHAOS Summary 2008", West Yarmouth, Mass., 2008.

[36] Standish Group. "Modernization – clearing a pathway to success", West Yarmouth, Mass., 2010.

Atomic Domains of Java Library Classes

Hisham M. Haddad, Woranuch Kaensaksiri, and Arjun Vasudevan
CSIS Department
Kennesaw State University
Kennesaw, Georgia 30144

Abstract - *Component reuse is evolving technology that is facing some challenges. One issue is the ease of reuse and the need for reuse solutions that facilitate self-selection of reusable components. This work builds on the "Atomic Domains and Wrappers" framework model that promotes highly reusable domain-specific software components. Through illustrative applications, the framework is applied to Java library classes to demonstrate the use of atomic domains to achieve effective component reuse. This work in an effort to facilitate software reuse and minimize coding effort when using library components organized into cohesive atomic domains. This work is a contribution to component-based development.*

Keywords: Software Reuse, Atomic Domains, Java Atomic Domains, Reusable Java Domains.

1 Introduction

Various approaches to the ease and effectiveness of component reuse [1,2,3] have been researched. This work attempts to further explore the application of the Atomic Domains framework [4,5,6,7] in developing reusable domain-specific components. The concept of atomic domains promotes component reuse. The general model of the Atomic Domains framework focuses on the developer's perspective of application development with reuse. The model facilitates effective domain-specific component reuse as applied by its wrapper. In this work, the components are source code, and the domain is Java applications using the Java library components. The application of the framework requires an effort to construct each atomic domain and its wrapper. The programming effort can vary. In this work, the effort is specific to the Java library components and is illustrated by sample Java-based atomic domains that are kept simple for proof of concept purpose. This work is a contribution to component-based software development that has become a defacto requirement for many in today's software industry.

2 Atomic Domain Framework

The Atomic Domain framework defines each atomic domain based on one related context. All reusable (and irreducible) components related to the context are conceptually placed in the same atomic domain [5]. Each atomic domain is associated with a wrapper, which includes a wrapper manager component. Application domain rules are built into the wrapper manager and are part of its construction process. The rules determine how the user application selects components from the atomic domain. Each atomic domain is associated with a command that is referenced in the application source code. The command refers to all components within the respective atomic domain. The command does not act as a single library call or an overloaded function. The atomic domain components are hidden from the developer and can only be referenced by the wrapper manager. When the command is used in the developer's source code, the compiler supplies all required data to the wrapper manager through the wrapper's API. The wrapper manager evaluates the command at the point it is used in the application source code and determines which component(s) to return to the application based on application domain rules [7].

3 Java Atomic Domains

The goal of this work is to apply the Atomic Domains framework model to the Java library classes and packages to facilitate effective reuse of library components in the development of Java applications, ease component management, and minimize coding effort. To achieve this goal, our approach is to develop illustrative atomic domains of related Java library components, and develop applications to demonstrate component reuse through the wrappers of developed atomic domains. Analysis of the Java library showed that the library includes a large number of related components (classes and packages). Sample applications selected for this work include the following:

Linear-Data Structure (DS) Atomic Domain: The DS atomic domain and its wrapper manager, called *DSWrapper*, are implemented as a class that includes data structure classes such as linked-list, stack, and vector. The purpose of wrapping reusable components into an atomic domain has been applied to data structure classes to minimize the design overhead and coding effort. To demonstrate the concept of wrapping existing components and retaining their original functionalities, the *DSWrapper* class defines a set of methods such as: add, clear, get, getSize, remove, and set as its API. This atomic domain is kept simple for illustration purposes.

Non-Linear Data Structure (NDS) Atomic Domain: The NDS atomic domain and its wrapper manager, called *NDSWrapper*, are implemented as a class that includes non-linear data structures such as Tree, Binary Tree, UGraph (un-weighted graph), and Abstract Graph. Like the DS atomic domain, wrapping reusable components into an atomic domain helps reduce coding effort and speeds up application development. The included components retain their original functionalities. The *NDSWrapper* class defines methods for Tree, Graph, and Binary Tree

structures. The Tree methods include createNode, deleteNode, setRoot, setLeave, create1stBranch, createBranch, and createTree; the Graph methods include createUgraph, addEdge, and getShortestPath; and the Binary Tree methods include search, insert, delete and getRoot. It should be noted that the linear and non-linear data structure components are separated into 2 atomic domains for illustration purposes. If such concepts are fully implemented in Java, many of the cumbersome design and coding of applications can be eliminated. From a programmer's perspective, this means less coding effort and fewer visible components to be accounted for.

GUI Atomic Domain: The GUI atomic domain and its wrapper, called *GUIWrapper*, include a number of GUI components such as button, checkboxes, dropdown lists, menu, and others. A GUI component, such as a button, can be associated with several classes. To build a button, the developer has to use a class to specify the color of the button, another class to apply a border to the button, and another class to specify the font style of the label or a class to specify an image picture for the button, and so on. This lengthy process can be repeated to build all the components that make up the application interface. With atomic domains, wrapping related components into an atomic domain can speed up the coding process significantly. With the GUI components (classes) being wrapped into one atomic domain, the application developer can use one reference to interact with the atomic domain (*GUIWrapper*) to build needed GUI components.

GUI components share common methods. For example, most GUI components define methods to change background and foreground colors, methods to modify label names, and others. The *GUIWrapper* (implemented as a class) incorporates most of these methods. This makes learning how to reuse GUI components easier by giving the user an abstract view of the *GUIWrapper* class with its methods that can provide the same functionalities of all the methods of many different GUI components. To motivate the wrapping of existing components and retaining their original functionalities, the *GUIWrapper* class defines methods such as: setBackgroundColor, setForgroundColor, setFont, setLabelName, setVisibility, isVisible, setEnabled, isEnable, setNewSize, getText, addActionListener, and addFocusListener. These methods provide the same functionalities as the individual methods.

Input/Output (IO) Atomic Domain: The IO atomic domain and its wrapper, called *IOWrapper* and is implemented as a class, includes a number of I/O classes such as File, ObjectInputStream, ObjectOutputStream, FileInputStream, FileOutputStream, BufferedReader, BufferedWriter, FileReader, FileWriter, DataInputStream, and DataOutputStream. Because of the nature of Java and in order to build an application with IO capabilities, the programmer has to use several library classes, such as the File class, to associate a file name and location with a File object during runtime. Also, InputStream and OutputStream classes are used to allow the application to read and write to a secondary storage device, and several more classes are used to handle the buffering process. With the *IOWrapper* and instead of having to decide how many classes and which one to use for IO process, the application developer can use one reference to the *IOWrapper* class and the wrapper manager creates necessary I/O components for the application. The listed IO components share common methods. For example, these IO components provide a link between the running program and a file object on an IO device. These IO components also have similar methods used to transfer data (read/write) to and from a file located on the IO device. IO classes do not have as many methods as the GUI classes because they provide a different type of service. To demonstrate the wrapping of existing components and retaining their original functionalities, the *IOWrapper* class defines methods such as readLine, read, writeFile, newLine, readFile, closeReaderStream, and closeWriterStream.

4 Illustrative Applications

Selected applications are described in this section to illustrate how Java atomic domains and their wrappers facilitate effective reuse of components in the development of Java applications. The approach is to contrast programming effort (lines of code) with and without using atomic domains. The selected applications are *Nine Tail*, *Tree*, *Payroll*, and *Bank Account*. These applications use the Java atomic domains described above.

Nine Tail Application: The *Nine Tail* application uses Breadth-First Search (BFS) algorithm to solve the Nail Tail problem. Nine coins are placed in a three by three matrix with some facing up and others facing down. A legal move is to take any coin that is facing up and reverse it, together with the coins adjacent to it. The task is to find the minimum number of moves that lead to all coins facing down. The problem is solved when all coins are facing down. This application utilizes the *NDSWrapper* class. Figures 1 and 2 show the source code before and after using the *NDSWrapper* class in the solution. The coding effort in Figure-2 is simpler than that in Figure-1.

Tree Application: The *Tree* application creates a folder structure and allows user to add, delete, and update values under a specified node. This application utilizes the *NDSWrapper* class. Figures 3 and 4 show the source code before and after using the NDS atomic domain. In Figure-3 the user needs to explicitly identify all the non-linear data structure components and their types while in Figure-4 the user creates all the NDS objects from the same wrapper class.

Bank Account Application: The *Bank Account* example illustrates the process of creating account objects and allowing the user to save account information into a file. This application utilizes the *GUIWrapper*, *IOWrapper*, and *DSWrapper* classes. Figures 5 and 6 show partial code before and after using the GUI, IO, and DS atomic domains. The omitted code in both Figures is identical. In Figure-5, the user needs to explicitly identify all required GUI, IO, and DS components and their types; while in Figure-6 the user creates all GUI, IO, and DS objects from respective same wrapper class. The coding effort in Figure-6 is simpler than that in Figure-5.

```
// Application Nine Tail without using the
// NDSWrapper classes

import java.util.*;
public class NineTailModel {
  private ArrayList<Node> nodes = new Array-
List<Node>(); // Vertices
  private ArrayList<AbstractGraph.Edge> edges =
    new ArrayList<AbstractGraph.Edge>(); // Store
edges
  private UnweightedGraph graph; // Define a
graph
  private AbstractGraph.Tree tree; // Define a
tree
  public NineTailModel() {
    createNodes(); // Create nodes
    createEdges(); // Create edges
    graph = new UnweightedGraph(edges, nodes);
    tree = graph.bfs(511);
  }
  private void createNodes() {
   for (int k1 = 0; k1 <= 1; k1++) {
    for (int k2 = 0; k2 <= 1; k2++) {
     for (int k3 = 0; k3 <= 1; k3++) {
      for (int k4 = 0; k4 <= 1; k4++) {
       for (int k5 = 0; k5 <= 1; k5++) {
        for (int k6 = 0; k6 <= 1; k6++) {
         for (int k7 = 0; k7 <= 1; k7++) {
          for (int k8 = 0; k8 <= 1; k8++) {
           for (int k9 = 0; k9 <= 1; k9++) {
                nodes.add(new Node(k1, k2, k3,
                     k4,k5, k6, k7, k8, k9));
    }  }  }  }  }  }  }  }  }
  public static class Node {
    int[][] matrix = new int[3][3];
    Node(int ...numbers) { // Variable-length
argument
      int k = 0;
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          matrix[i][j] = numbers[k++];
        }
      }
    }
    Node(int[][] numbers) {
      this.matrix = numbers;
    }
    public boolean equals(Object o) {
      int[][] anotherMatrix = ((Node)o).matrix;
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          if (matrix[i][j] != anotherMa-
trix[i][j]) {
            return false;
          }
        }
      }
      return true; // Nodes with the same matrix
values
    }
```

```
public String toString() {
      StringBuilder result = new StringBuilder();
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          result.append(matrix[i][j] + " ");
        }
        result.append("\n");
      }
      return result.toString();
    }
  }
  private void createEdges() {
    for (Node node : nodes) {
      int u = nodes.indexOf(node); // node index
      int[][] matrix = node.matrix; // matrix for the
node
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          if (matrix[i][j] == 0) {
            Node adjacentNode =
getAdjacentNode(matrix, i, j);
            int v = nodes.indexOf(adjacentNode);
            edges.add(new AbstractGraph.Edge(v, u));
 }  }  }  }  }

  private Node getAdjacentNode(int[][] matrix, int i,
int j) {
    int[][] matrixOfNextNode = new int[3][3];
    for (int i1 = 0; i1 < 3; i1++) {
      for (int j1 = 0; j1 < 3; j1++) {
        matrixOfNextNode[i1][j1] = matrix[i1][j1];
      }
    }
    flipACell(matrixOfNextNode, i - 1, j);// Top
    flipACell(matrixOfNextNode, i + 1,j); // Bottom
    flipACell(matrixOfNextNode, i, j - 1);// Left
    flipACell(matrixOfNextNode, i, j + 1);// Right
    flipACell(matrixOfNextNode, i, j);    // Self
    return new Node(matrixOfNextNode);
  }
    private void flipACell(int[][] matrix, int i, int
j) {
    if (i >= 0 && i <= 2 && j >= 0 && j <= 2) { //
Within boundary
      if (matrix[i][j] == 0) {
        matrix[i][j] = 1; // Flip from 0 to 1
      }
      else {
        matrix[i][j] = 0; // Flip from 1 to 0
 }  } }
  public LinkedList<Node> getShortestPath(Node node)
{
    Iterator iterator =
tree.pathIterator(nodes.indexOf(node));
    LinkedList list = new LinkedList();
    while (iterator.hasNext())
      list.addFirst(iterator.next());
    return list;
 }  }
```

Figure-1: Application Nine Tail without using the *NDSWrapper* Class.

Payroll Application: The *Payroll* application calculates the check amount by allowing the user to enter the hours worked and the pay rate. The application allows the user to save the check amount to a text file. This application uses the *GUI-Wrapper* and *IOWrapper* classes to create the user interface and to add saving to file capability. Without atomic domains, the user needs to explicitly identify all required GUI components and their types; while with atomic domains the user creates all GUI objects from the same wrapper classes. Coding with the later approach is simpler than with the former approach.

```
// Application Nine Tail with using the
// NDSWrapper classes

import java.util.*;
public class NineTailModel {
  private NDSWrapper ugraph  = new NDSWrap-
per("ugraph");;
 private ArrayList<NDSWrapper.Node> nodes = new
ArrayList<NDSWrapper.Node>(); // Vertices
   public NineTailModel() {
    ugraph = new NDSWrapper("ugraph");
    createNodes(); // Create nodes
    createEdges(); // Create edges
    ugraph.createUgraph(nodes);
  }
  private void createNodes() {
    for (int k1 = 0; k1 <= 1; k1++) {
     for (int k2 = 0; k2 <= 1; k2++) {
      for (int k3 = 0; k3 <= 1; k3++) {
       for (int k4 = 0; k4 <= 1; k4++) {
        for (int k5 = 0; k5 <= 1; k5++) {
         for (int k6 = 0; k6 <= 1; k6++) {
          for (int k7 = 0; k7 <= 1; k7++) {
           for (int k8 = 0; k8 <= 1; k8++) {
            for (int k9 = 0; k9 <= 1; k9++) {
                 nodes.add(new NDSWrapper.Node(k1,
                 k2, k3, k4,k5, k6, k7, k8, k9));
    } }  } } }  } } } } }

  private void createEdges() {
    for (NDSWrapper.Node node : nodes) {
      int u = nodes.indexOf(node); // node index
      int[][] matrix = node.matrix; // matrix for
the node
      for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
          if (matrix[i][j] == 0) {
            NDSWrapper.Node adjacentNode =
getAdjacentNode(matrix, i, j);
            int v = nodes.indexOf(adjacentNode);
            ugraph.addEdge(v, u);
      } } } } }
```

```
private NDSWrapper.Node getAdjacentNode(int[][] ma-
trix, int i, int j) {
    int[][] matrixOfNextNode = new int[3][3];
    for (int i1 = 0; i1 < 3; i1++) {
      for (int j1 = 0; j1 < 3; j1++) {
        matrixOfNextNode[i1][j1] = matrix[i1][j1];
      }
    }
    flipACell(matrixOfNextNode, i - 1, j); // Top
neighbor
    flipACell(matrixOfNextNode, i + 1, j); // Bottom
neighbor
    flipACell(matrixOfNextNode, i, j - 1); // Left
neighbor
    flipACell(matrixOfNextNode, i, j + 1); // Right
neighbor
    flipACell(matrixOfNextNode, i, j);     // Flip
self
    return new NDSWrapper.Node(matrixOfNextNode);
  }
  private void flipACell(int[][] matrix, int i, int
j) {
    if (i >= 0 && i <= 2 && j >= 0 && j <= 2) { //
Within boundary
      if (matrix[i][j] == 0) {
        matrix[i][j] = 1; // Flip from 0 to 1
      }
      else {
        matrix[i][j] = 0; // Flip from 1 to 0
      }
    }
  }
  public LinkedList<NDSWrapper.Node> getShortest-
Path(NDSWrapper.Node node){
      LinkedList list = new LinkedList();
      list = ugraph.getShortestPath(nodes,node);
      return list;
  }
}
```

Figure-2: Application Nine Tail using the *NDSWrapper* Class.

5 Conclusion

The implementation of component reuse ranges from isolated solution such as using new languages [8], wrapping legacy components [9], refactoring [10], program mining [11], product line technologies [12,13], and product populations [14] to the use of current industry standards for distributed components. This work supports the local component model, where a component is defined as Java source code of a module, class, function, or code snippet. The presented applications focus on Java classes as components. The components' atomic domains group similar and related Java components that are bound with a wrapper, and the domain is utilized via a single atomic reference in the application source code. These applications illustrate ease of reuse as well as encourage a higher amount of reuse of Java library components.

The initial outcomes from this work indicate promising results. The example applications illustrate the possibility of achieving effective reuse of Java components and that the programming effort for the application developer can be made easier. Instead of requiring detailed knowledge of every library component and package and their methods, with this approach the application developer needs to know about smaller number of components (atomic domains) and their methods. From lines of code perspective, some atomic domains can reduce source code significantly, such as in the case of the Payroll and Bank Account applications. In other cases, such as the GUI atomic domain, the user creates many objects (components) from a single atomic domain without the need to know the internal details for each created object. During the development we found that using atomic domains and their wrappers is easier to get the applications developed than not using these wrapper classes. In addition to having to deal with few wrapper classes, the management overhead of such classes was improved compared to dealing with the many classes in the library.

Future work will focus on identifying more related components in the Java library, developing more concrete atomic domains and their wrappers; developing more illustrative applications, and studying improvement in coding effort (with vs. without wrapper classes).

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.GraphicsEnvironment;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JRootPane;
import javax.swing.JTree;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.DefaultTreeCellRenderer;
import javax.swing.tree.DefaultTreeModel;
public class TreeWrap extends JFrame {
    private JTree tree;
    private String nodename, parentname, rootname;
    private DefaultTreeModel model;
    private DefaultMutableTreeNode nNode;
    private DefaultMutableTreeNode dNode;
    private DefaultMutableTreeNode parent, outdoor,
indoor,ski, soccer, tennis, pingpong, iceskate, chess;
    private String [] sport = {"Outdoor
Sport","Ski","Soccer","Tennis",
                               "Indoor
Sport","Pingpong","Ice skate","Chess"};
    ReadImage img, oimg, cimg;
        Image i1, i2, i3;
        ImageIcon licon, oicon, cicon;
        DefaultTreeCellRenderer renderer, renderer2;
        public TreeWrap(int indexicon){
          parent = new DefaultMutableTreeNode("Sport",
true);
    outdoor = new DefaultMutableTreeNode("Outdoor
Sport");
    indoor = new DefaultMutableTreeNode("Indoor
Sport");
    ski = new DefaultMutableTreeNode("Ski");
    soccer = new DefaultMutableTreeNode("Soccer");
    tennis = new DefaultMutableTreeNode("Tennis");
    pingpong = new DefaultMutableTreeNode("Pingpong");
    iceskate = new DefaultMutableTreeNode("Ice
Skate");
    chess = new DefaultMutableTreeNode("Chess");
    parent.add(outdoor);
    parent.add(indoor);
    outdoor.add(ski);
    outdoor.add(soccer);
    outdoor.add(tennis);
    indoor.add(pingpong);
    indoor.add(iceskate);
    indoor.add(chess);
    tree = new JTree(parent);
     setCustomIcon();
     setNoIcon();
     setStandardIcon();
     if (indexicon == 1){
        tree.setCellRenderer(renderer); // custom icon
      }else if(indexicon ==2){
         tree.setCellRenderer(renderer2); // no icon
      }
              tree.setBackground(Color.PINK);
               setTitle("Sample tree");
        setDefaultCloseOpera-
tion(JFrame.EXIT_ON_CLOSE);
        Dimension d = new Dimension(600, 600);
        setSize(d.width, d.height);
        setLayout(new BorderLayout());
        add(tree, BorderLayout.CENTER);
        setUndecorated(true);
        getRoot-
Pane().setWindowDecorationStyle(JRootPane.PLAIN_DIALOG
);
        setVisible(true);
    }
```

```
public void setFont(int index, int size){
      GraphicsEnvironment gEnv = GraphicsEnviron-
ment.getLocalGraphicsEnvironment();
       String envfonts[] =
gEnv.getAvailableFontFamilyNames();
       tree.setFont(new Font(envfonts[index],
Font.BOLD,size));
  }
    public void AddNode(String nodename, String parent-
name){  model = (DefaultTreeModel)tree.getModel();
        nNode = new DefaultMutableTreeNode(nodename);
        if (parentname == "Sport"){
            model.insertNodeInto(nNode, parent, par-
ent.getChildCount());
        }else if (parentname == "Outdoor Sport"){
            model.insertNodeInto(nNode, outdoor, out-
door.getChildCount());
        }else if (parentname == "Indoor Sport"){
            model.insertNodeInto(nNode, indoor, in-
door.getChildCount());
        }else if (parentname == "Ski"){
            model.insertNodeInto(nNode, ski,
ski.getChildCount());
        }else if (parentname == "Soccer"){
            model.insertNodeInto(nNode, soccer, soc-
cer.getChildCount());
        }else if (parentname == "Tennis"){
            model.insertNodeInto(nNode, tennis, ten-
nis.getChildCount());
        }else if (parentname == "Pingpong"){
            model.insertNodeInto(nNode, pingpong, ping-
pong.getChildCount());
        }else if (parentname == "Ice Skate"){
            model.insertNodeInto(nNode, iceskate,
iceskate.getChildCount());
        }else if (parentname == "Chess"){
            model.insertNodeInto(nNode, chess,
chess.getChildCount());
        }  }
    public void deleteNode(String nodename){
      model = (DefaultTreeModel) (tree.getModel());
      if (nodename == "Sport"){
          model.removeNodeFromParent(parent);
      }else if (nodename == "Outdoor Sport"){
          model.removeNodeFromParent(outdoor);
      }else if (nodename == "Indoor Sport"){
          model.removeNodeFromParent(indoor);
      }else if (nodename == "Ski"){
          model.removeNodeFromParent(ski);
      }else if (nodename == "Soccer"){
          model.removeNodeFromParent(soccer);
      }else if (nodename == "Tennis"){
          model.removeNodeFromParent(tennis);
      }else if (nodename == "Pingpong"){
          model.removeNodeFromParent(pingpong);
      }else if (nodename == "Ice Skate"){
          model.removeNodeFromParent(iceskate);
      }else if (nodename == "Chess"){
          model.removeNodeFromParent(chess);
      } }
    public String[] getSport() { return sport; }
    public void setStandardIcon(){}
    public void setNoIcon(){
      renderer2 = new DefaultTreeCellRenderer();
      renderer2.setOpenIcon(null);
      renderer2.setClosedIcon(null);
      renderer2.setLeafIcon(null);
}
 public void setCustomIcon(){
    img = new ReadImage(0); oimg = new ReadImage(1);
    cimg = new ReadImage(2);
    i1 = img.getImage();  i2 = oimg.getImage();
    i3 = cimg.getImage();
    licon = new ImageIcon(i1);  oicon = new ImageIcon(i2);
    cicon = new ImageIcon(i3);
    renderer = new DefaultTreeCellRenderer();
    renderer.setOpenIcon(oicon);
    renderer.setClosedIcon(cicon);
    renderer.setLeafIcon(licon);
} }
```

Figure-3: Application Tree without using *NDSWrapper* class.

```
// Application Tree with using the
// NDSWrapper classes

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import javax.swing.JFrame;
import javax.swing.JRootPane;
import trees.NDSWrapper;
public class TreeWrap extends JFrame {
    private NDSWrapper wtree;
    private String [] sport = {"Outdoor
Sport","Ski","Soccer","Tennis",
                               "Indoor
Sport","Pingpong","Ice skate","Chess"};
    private String nodename,parentname;
    public TreeWrap(int indexicon){
        nodename = "Special";
        parentname ="Sport";
        wtree = new NDSWrap-
per("tree","Sport",sport);
        wtree.create1stBranch(0);
        wtree.create1stBranch(4);
        wtree.createBranch(0,1);
        wtree.createBranch(0,2);
        wtree.createBranch(0,3);
        wtree.createBranch(4,5);
        wtree.createBranch(4,6);
        wtree.createBranch(4,7);
        wtree.createTree(indexicon);
        wtree.setColor();
```

```
setTitle("Sample tree");
        setDefaultCloseOpera-
tion(JFrame.EXIT_ON_CLOSE);
        Dimension d = new Dimension(600, 600);
        setSize(d.width, d.height);
        setLayout(new BorderLayout());
        add(wtree, BorderLayout.CENTER);
        setUndecorated(true);
        getRoot-
Pane().setWindowDecorationStyle(JRootPane.PLAIN_DI
ALOG);
        setVisible(true);
    }
    public void AddNode(String nodename, String
parentname){
         this.nodename = nodename;
        this.parentname = parentname;
        wtree.createNode(nodename,parentname);
    }
    public void DeleteNode(int nodeindex){
        wtree.deleteNode(nodeindex);
    }
    public void SetFontNode(int fontfamilyIndex,
int fontsize){
        wtree.setFont(fontfamilyIndex, fontsize);
    }
    public String[] getSport() {
        return sport;
    }
}
```

Figure-4: Application Tree using the *NDSWrapper* Class.

```
// Application BankAccounts without using the
// GUI, IO, and DS Wrapper classes
import java.io.Serializable;
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BankAccounts extends JFrame implements Ac-
tionListener{
// This section declares the GUI & Data Structure
// components needed for this program.
  JTextField jAccountNumber, jBalance;
  JLabel accountNumber, balance;
  JButton bSave;
  JTextArea display;
  JScrollPane sp_display;
  LinkedList accountList;
  Account tempAccount = new Account();

  public BankAccounts() {
     BankAccountsLayout customLayout = new
         BankAccountsLayout();
     getContentPane().setFont(new
        Font("Helvetica", Font.PLAIN, 12));
     getContentPane().setLayout(customLayout);

     // This section create the GUI & Data
     // Structure objects.

     jAccountNumber = new JTextField();
     accountNumber  = new JLabel();
     jBalance       = new JTextField();
     balance        = new JLabel();
     bSave          = new JButton();
     display        = new JTextArea();
     accountList    = new LinkedList();
     // the rest of the code is omitted. It is
     //identical in both versions.
```

```
  // This section enables write/read
  // capabilities without using wrappers.

  public void writeAndRead() {
   try
    { File file = new File("bankAccounts.dat");
      if(!file.exists()){
        boolean c = file.createNewFile();
    }

    FileOutputStream fileOutputStream = new
         FileOutputStream(file);
    ObjectOutputStream objectOutputStream = new
         ObjectOutputStream(fileOutputStream);
    objectOutputStream.writeObject(accountList);
    objectOutputStream.flush();
    objectOutputStream.close();
    ObjectInputStream objectInputStream = new
         ObjectInputStream(new
                     FileInputStream(file));
    AccountList = (LinkedList)
                  objectInputStream.readObject();
    objectInputStream.close();
   }

   catch(ClassNotFoundException cnfexception)
    { System.err.println
      ("ClassNotFoundException "); }

   catch(IOException ioexception)
    { System.err.println("IOException ");}
    }

// the rest of the code is omitted as it is
// identical in both versions of the application.
```

Figure-5: Application Bank Account without using the *GUIWrapper*, *IOWrapper*, and *DSWrapper* classes.

```
// Application BankAccounts using the GUI, IO, and
// DS Wrapper classes

import java.io.Serializable;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class BankAccountsWithWrappers extends
       JFrame implements ActionListener{

// This section declares the GUI & Data
// Structure components needed for this
//  program.

    GuiWrapper jAccountNumber, jBalance,
               accountNumber, balance, bSave,
               display;
    JScrollPane sp_display;
    DSWrapper accountList;
    Account tempAccount = new Account();
    public BankAccountsWithWrappers() {
       BankAccountsLayout customLayout = new
          BankAccountsLayout();
       getContentPane().setFont(new
          Font("Helvetica", Font.PLAIN, 12));
       getContentPane().setLayout
          (customLayout);
```

```
       // This section create the GUI & DS objects.

       jAccountNumber = new GuiWrapper("textfield");
       accountNumber  = new GuiWrapper("label");
       jBalance       = new GuiWrapper("textfield");

       balance = new GuiWrapper("label");
       bSave   = new GuiWrapper("button");
       display = new GuiWrapper("textarea");
       accountList = new DSWrapper("linkedlist");

// This section utilizes the IOWrapper class
// for write/read purposes.

public void writeAndRead() {
   IOWrapper ioWrapper = new
     IOWrapper("objectio", "bankAccounts.dat");

   ioWrapper.write(accountList);
   ioWrapper.closeWriterStream();

   accountList.linkedList =
        LinkedList)ioWrapper.readObject();

   ioWrapper.closeReaderStream();
}

// the rest of the code is omitted as it is
// identical in both versions of the application.
```

Figure-6: Application Bank Account using the *GUIWrapper*, *IOWrapper*, and *DSWrapper* classes.

6 REFERENCES

[1] T. Khammaci, M. Oussalah, and B. Giuseppin. *Towards a reuse-based pattern representation and search model*. Proceedings of the International Conference of Software Engineering Research and Engineering (SERP'02), Jun 24-27, 2002, SCREA Press, pp. 267-273.

[2] B. Councill and G.T. Heineman, G.T. *Component based software engineering and issue of trust.* Proceedings of the International Conference on Software Engineering (ICSE'00), Limerick, Ireland, 661-664.

[3] F. Bachman, L. Bass, et. al. *Technical Concepts of Component-Based Software Engineering*. Technical Report, CMU/SEI-2000-TR-008, ESC-TR-2000-007.

[4] Mauricio Ordoñez and Hisham M. Haddad. *Enhanced Component Reuse with Atomic Domains: Application Scenarios*. Proceedings of the IEEE International Conference on Information Technology: New Generations (ITNG'07), Las Vegas, April 2007, pp. 597-602.

[5] Hisham Haddad and Ying Xie. *Wrapper-Based Framework for Domain-Specific Software Reuse*. Journal of Information Science and Engineering (JISE), Vol. 22, No. 2, March 2006, pp. 269-282.

[6] Hisham Haddad and Walter Fortner. *A Framework for Domain-Specific Reusable Components*. Proceedings of the International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, June 2003, pp. 384-390.

[7] Hisham Haddad and Herbert Tesser. *Reusable Subsystems: Domain-Based Approach.* Proceedings of the Annual ACM Symposium on Applied Computing, Madrid, Spain, March 2002.

[8] Vugranam C. Sreedhar. Mixin'Up Components. ICSE'02, Orlando, Florida, May 2002, pp. 198-207.

[9] Jiang Guo and Yuehong Liao. Integrating Software Components through Wrapper Technologies. Proceedings of the 7'th IASTED International Conference, Software Engineering and Applications, Marina Del Rey, CA, November 2003, pp. 441-446.

[10] Rodrigo E. Caballero and Steven A. Demurjian Sr., Towards the Formalization of a Reusability Framework for Refactoring, ICSR-7, LNCS-2319, 2002, pp. 293-308.

[11] Donglin Xia, Yaoxue Zhang and Ling Zhou, A Multi-Agent System for Program Mining, Proceedings of the 7'th IASTED International Conference, Software Engineering and Applications, Marina Del Rey, CA, November 2003, pp. 95-99.

[12] Kwanwoo Lee, Kyo C. Kang and Jaejoon Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering, ICSR-7, LNCS-2319, 2002, pp. 62-77.

[13] Colin Atkinson and Dirk Muthig, Enhancing Component Reusability through Product Line Technology, ICSR-7, LNCS-2319, 2002, pp. 93-108.

[14] Rob van Ommering, Building Product Populations with Software Components, ICSE'02, Orlando, Florida, May 2002, pp. 255-265.

Modeling Virtual Machine Packing Factor using a Third Party Tool and other Strategies

Dr. Carl J. De Pasquale

Abstract - Restrain datacenter growth, reduce physical server deployment, decrease power/cooling requirements, go green and do more with less, clichés I'll admit, but familiar to be sure. Organizations struggling with increasing IT expenses are seeking ways to reduce operating costs while improving service levels. For many organizations virtualization is strategic. Virtualization helps reduce costs by consolidating lightly utilized servers onto high performance, power efficient multi socket/multi core platform. Great, but how many lightly utilized physical servers (or applications residing on lightly utilized physical servers) can be or consolidated onto a hypervisor based hardware? This paper presents an approach to estimate maximum number of VM that should be packed onto a host (hypervisor) system.

Key Words: Modeling, Virtualization, Sizing, Migration, Trending and Tracking

I. Introduction

Historical

To be sure virtualization is not a new technology. Originally marketed by IBM during the late 1960's and early 1970's as Virtual Machine/Conversational Monitor System (VM/CMS), VM/CMS support simultaneous execution of multiple operating systems on a single mainframe [1]. Sound familiar. VM/CMS was once heavily used by universities and fortune 500 corporations, only to be displaced during the advent of PCs. After making a brief resurgence during the Y2K era, VM/CMS faded back into obscurity, only to arise like a phoenix, in the guise of VMWare™, ZOS/ZLinux™, et al.

Virtualization Goal

Current virtualization solutions attempt to reduce the number of lightly utilized physical servers by migrating the operating system and application software onto power efficient and highly utilized multi-socket/multi-core systems.

The benefits can be enormous; restrain datacenter growth, slow physical server deployment, decrease power/cooling requirements, redesign and improve disaster recovery scenarios and all while maintaining (and possibly improving) application performance (SLA).

The remainder of this paper is organized into V sections which virtualization. Section II discusses high level virtualization planning tasks. Section III illustrates the use of BMC[1] to model a P2V migration. Section IV presents a SPECmark based strategy to size virtual hosts. Section V discusses managing the virtual datacenter and section VI provides brief commentary on the overall process.

II. P2V Process Flow

Application Selection

Figure 1, swim lanes 1 and 2 show that a P2V migration begins by choosing an application whose performance characteristics favor virtualization. Generally we seek application(s) exhibiting low CPU usage and small memory footprint. To identify these applications the following key performance indicators (KPI) are gathered:

- *Application Peak periods – hour(s), day(s), month(s), seasonal information*
- *Physical server specification,* number and type of servers, CPU speed and SPECmark rating
- Installed vs. used physical server capacity of CPU and memory, I/O disk and network

Once the KPI (or application characterization) data are available the number of "New Host Systems (hypervisor)" and VM packing factor must be selected and comfigured. The ultimate configuration is defined using a modeling tool, SPECmark[2] data or both. The details of each strategy is outlined in swim lane 3 and discussed in sections III and IV.

The 4th swim lane provides an opportunity to test and tune the new virtual configuration. This task is highly recommended, especially for initial migrations as it provides an opportunity to verify the VM configuration

[1] BMC performance assurance is used as an example software product. There are several other tools from various vendors such as Hyperformix, PerfCap, Metron, TeamQuest that provide similar functionality and any of these tools can be used.

[2] Section 3 and 4 discuss the approach.

and VM placement strategy prior to go live. Lastly, swim lane 5 suggests that virtual datacenters must be actively monitored and managed. As P2V migrations progress, virtual datacenter resources such as "New Host Servers", SAN disk, and network bandwidth are consumed.

The timely replenishment of virtual datacenter resources is critical because for many organizations the procurement cycle can take months. If hardware resources become scarce, application performance may degrade and continued virtualization projects could be delayed.

III. Model It – How many VMs?

Figure 2 shows that 8-physical servers have been selected for virtualization. We begin by using the "Analyze" facility of the BMC tool to create an application model. The model created by the BMC tool is based on the work of [2].

Since virtualization supports simultaneous execution of multiple operating systems on a single "New Host System (hypervisor)" each candidate physical server will be modeled as a single consolidated virtual workload.

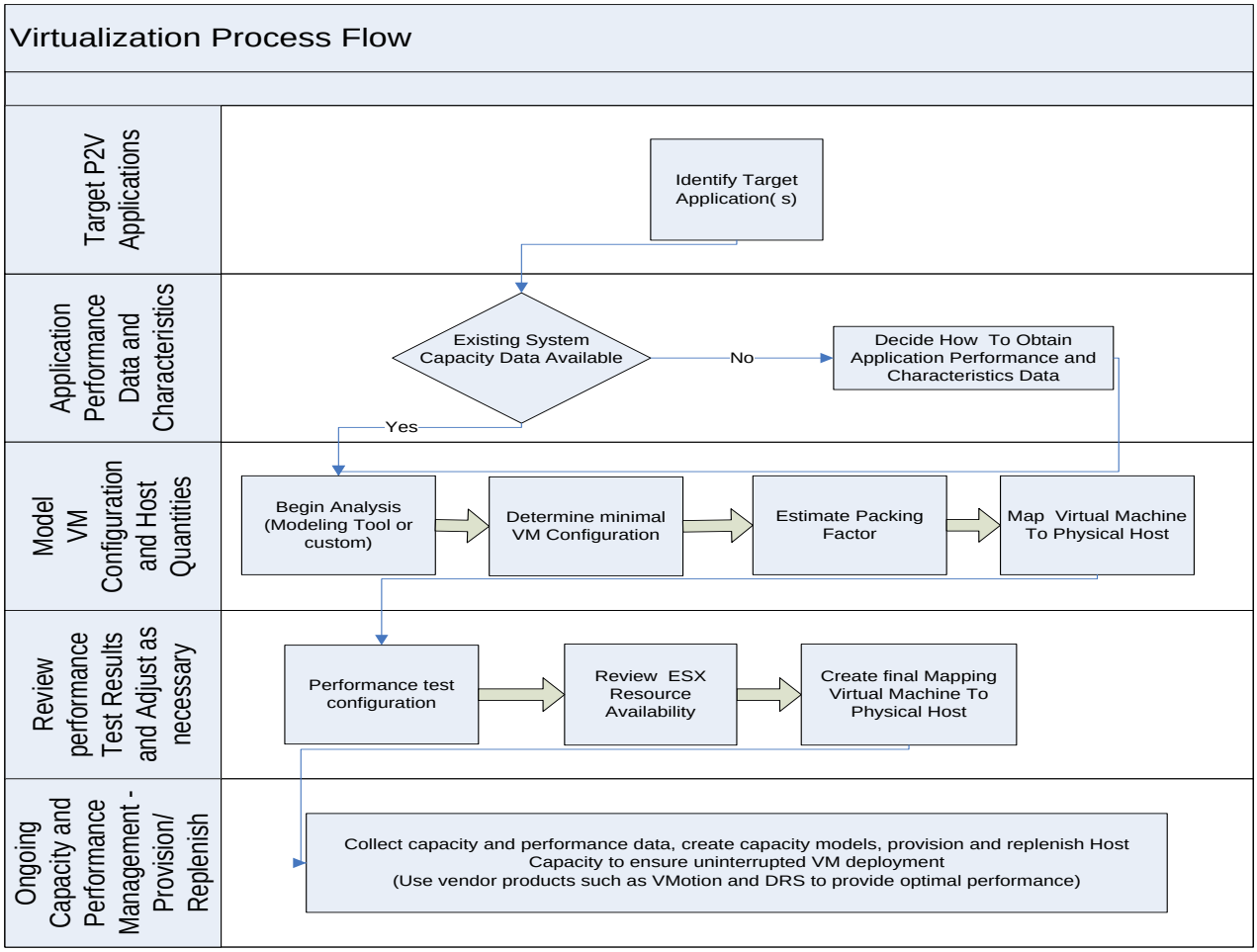


Figure 1: Virtualization Process Flow

To create a single consolidated virtual workload we delete all previously defined workloads (non-zzz[3]). By deleting all predefined workload the applications resources are combined into a single default defined by the BMC product as zzz, see Figure 3. The model with the single zzz workload is saved and rerun using "Analyze" The single workload (zzz) is now representative of a single VM. Next we

[3] zzz is a default workload. All process not allocated to a specific workload to will be assigned to zzz

need to determine how many VMs will fit on a"New VMware Host System (Hypervisor)[4]".

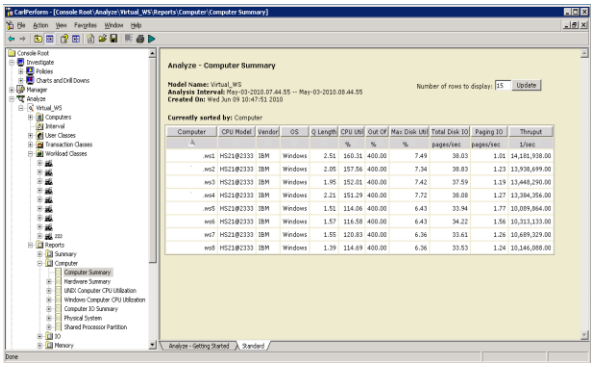


Figure 2: Multiple Workloads

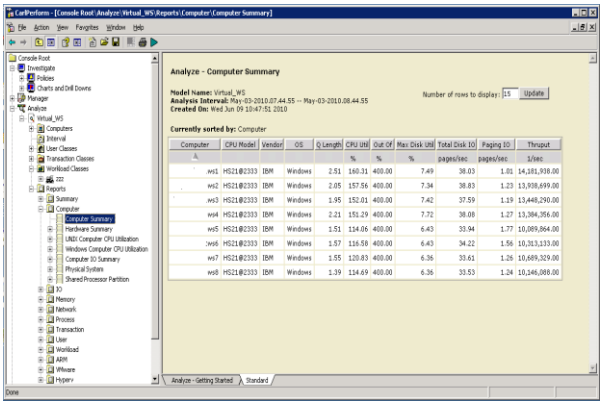


Figure 3: Single ZZZ Workload

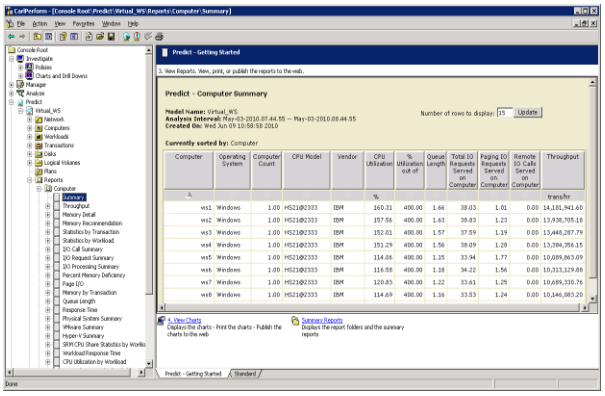


Figure 4: Predict Server Resources

Initial Predict Model

With the workloads (zzz) now representing a VM, the "Predict" component is used to create the "New VMware Host System (Hypervisor)" upon which the VMs will run.

As shown in Figure 4 the "Predict" model is opened and evaluated by right clicking on the model name "Virtual_WS" and selecting "evaluate". The resulting computer summary must match the analyze computer summary shown in Figure 3.

Create VMWare Host System

To create a "New VMware Host System (Hypervisor)" to run the VMs, right click on the computer and select "New VMware Host System". Choose a physical server from the hardware table shown in Figure 5. If the server is not listed, download the latest hardware table or enter the SPECmark information in a user defined hardware table entry. Setting the memory and CPU thresholds as shown in Figure 6 and Figure 7 complete the creation of the "New Virtual Host System".

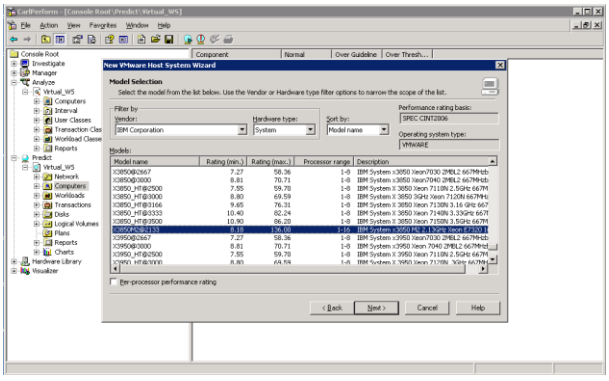


Figure 5: Select Physical Host Model Type

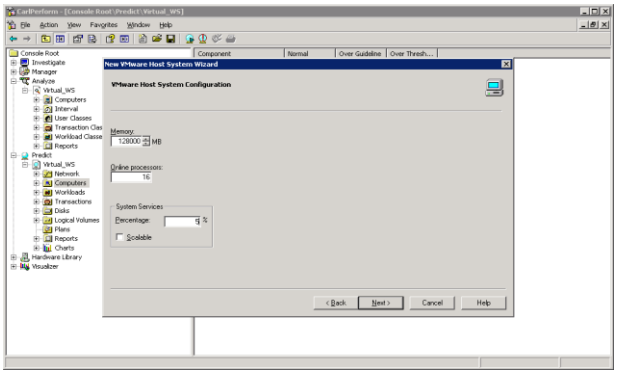


Figure 6: Define Physical Host Memory

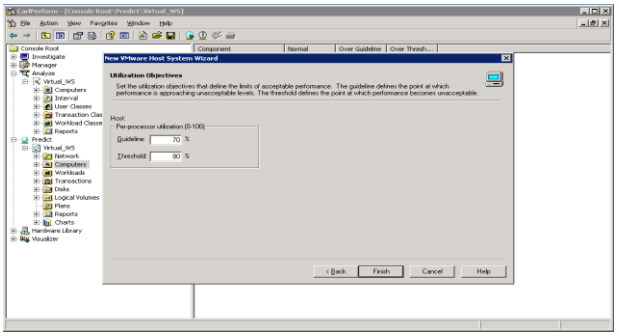


Figure 7: Set Threshold

[4] The organization has standardized on a "New VMware Host System (Hypervisor)

Estimating VM Resources

Modeling a VM is simple. Memory and the number of vCpus are the only requirements. The amount of memory used by the application is easily be obtained from a modeling tool's data collector/analyzer, ps, top or perfmon. For production applications do not over commit memory. If you do and memory ballooning occurs response time will substantially degraded. Estimating vCpus is trickery. To estimate the number of vCpus needed we convert the physical CPU utilization to a virtual estimate. The BMC tool represents CPU utilization as 100 per CPU. So a 4 CPU system will have a utilization of 400. The estimation process begins by reviewing the CPU KPIs. Web Server 1 consumed 160 out of 400. The source server 2.33 GHz and the "New Host System" is 2.13 GHz, 160 adjusted to the target host is roughly 175. This approach assumes that source and target chip architecture share a common supported baseline [3]. We assume a 5% "New Host System" overhead a 2% per VM overhead which gives and adjusted CPU of roughly 212. These estimates are empirical and may differ depending on environment or server configuration. The number of vCpus is given by equation 1, well almost.

$$3 = Ceil(\frac{212}{400}) \qquad (1)$$

Create VMs

The newly created physical server and its VMs are shown in Figure 8, Figure 9 and Figure 10. Right clicking on the "New Host System" and selecting New Virtual machine creates VMs. The VM name and OS type UNIX or Windows are selectable options. Figure 9 shows where vCpus and memory are entered. Notice that the number of vCpus is 2 and that is obtained from a purely MHz rating. Referring to Figure 3 less than two physical (1.6) CPUs were required. Why then should we specify 4 instead of 2? MHz rating is misleading. MHz do not account for concurrent threads of execution during each scheduling interval. Highly threaded applications will suffer performance bottlenecks when the number of available vCPU per schedule is too low. By equation 1 and by understanding the application behavior we confidently allocate 3 vCpus. But wait, 4 vCpus are allocated. Why? The vendor recommends allocation in pairs. Each VM is created using the same strategy. Figure 10 shows eight (8) VMs. VMs1 through 4 are 4 vCpus VMs and VMs 5 through 8 are 2 vCpus.

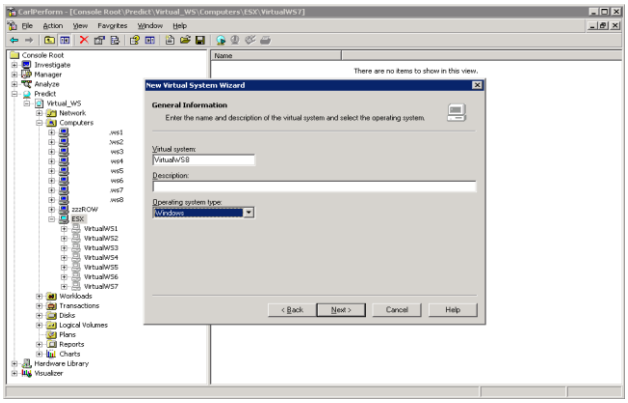


Figure 8: Create VM

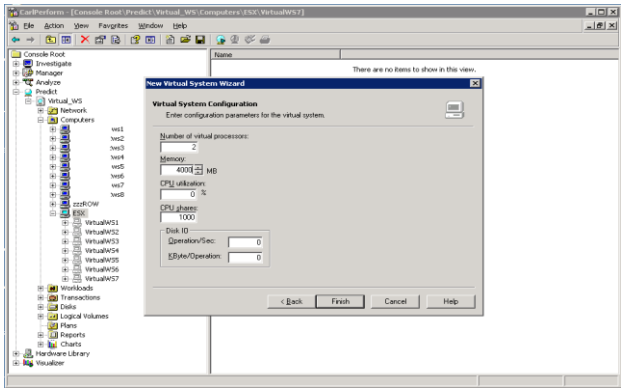


Figure 9: Setting VM vCpu and Memory

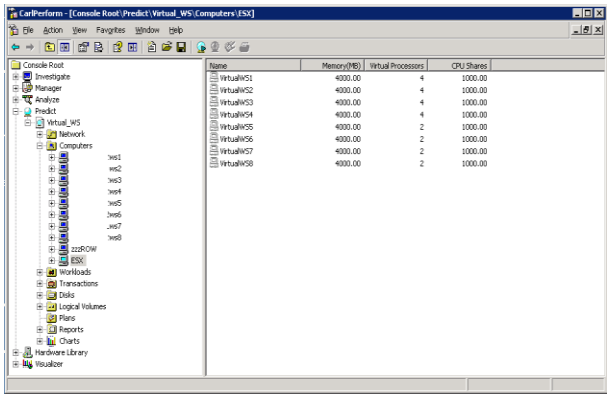


Figure 10: Eight (8) Physical, Eight (8) VMs

Running the Virtual Model

Once the "New Host Server (hypervisor)" and VMs are created, the zzz workloads created in analyze and carried forward to predict must now be migrated to the VMs. To migrate a physical zzz to a VM, right click on workload, check the physical server, click on the VM and the click add, see Figure 11.

Once all physical server's zzz workload are migrated to a VM the predict model is run by right clicking on the model name and selecting, evaluate.

As seen in Figure 12 these eight (8) web servers will not comfortably fit on the single "New Host System

(Hypervisor)". The "New Host System" is running over 90% and each VM is waiting nearly 25%. To reduce the waiting time another "Host System", CapacityESX, see Figure 13, is created and VirtualWS1 is moved to it.

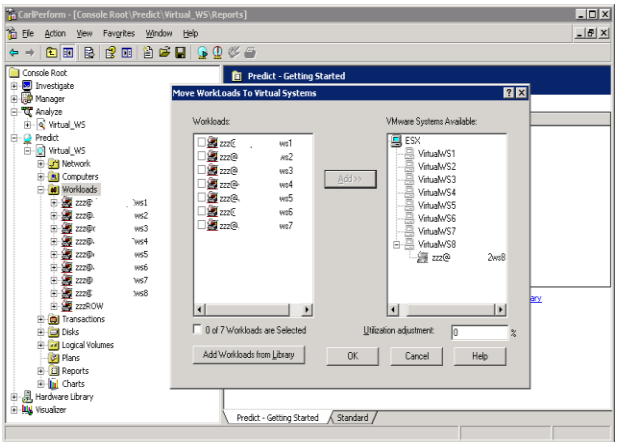


Figure 11: Workload Migration

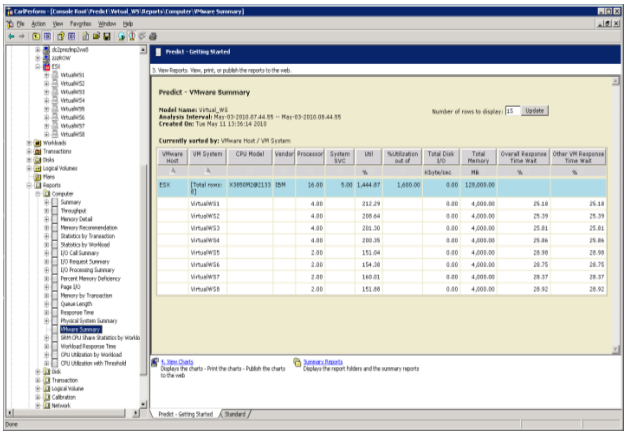


Figure 12: Resource Estimate

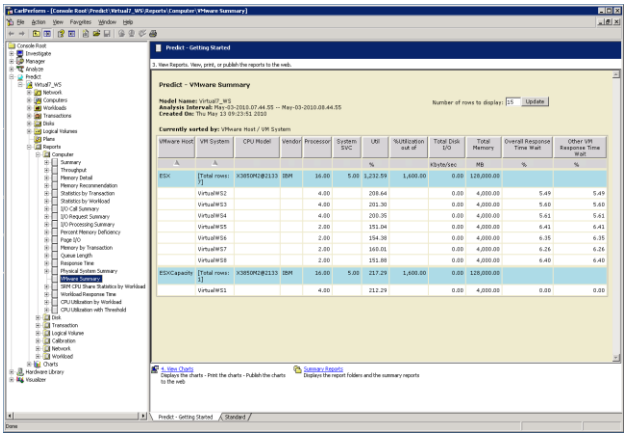


Figure 13: Improving Wait Time

When a single workload is moved to CapacityESX, CPU utilization is roughly 77% and the waiting time drops to roughly 5.5 – 6.5%. Migrating two VMs, to CapacityESX, leaving 6 VMs on ESX, provides Host utilization of roughly 67% on ESX and the waiting of roughly 2%.

IV. Estimate VMs using SPECmark

When modeling tools are not available or even when they are, the approximate number of VMs that can be consolidated onto a "New Host Server (hypervisor)" can be estimated using SPECmark data. The approach has a pre-condition that the source and target chip architecture share a common supported baseline [3] [4]. As an example we will use Cint2006 published by SpecOrg [5] for the source and target hosts. The source host has a Cint2006 rating of 32.9 and our application is running on peak 40% CPU utilization. The target host has a Cint2006 rating of 143. Using these performance ratings we can estimate how many physical servers (with similar CPU profiles) can be migrated to a virtual host. Equation 2 shows the adjusted CPU utilization. Equation 3 normalizes the existing server utilization to the target host at 70%. Finally, the number of VMs that may be able to run on the "Host System" is given in Equation 4. The calculations shown uses a peak server utilization of 40%, see Figure 2. Since all eight server are not running at a peak of 40%, the calculations, as expected, result in slightly smaller packing factor than the predict model, 6 vs. 7. See Figure 13. Using the average CPU utilization of 34% the packing factor is now in line with the analytical model.

$$11.53 = 40 * (\frac{45}{156}) \quad (2)$$

$$947 = 156 / \frac{11.53}{70.00} \quad (3)$$

$$6 = \text{int}(\frac{947}{156}) \quad (4)$$

V. Manage It - Maintain Resources

Designing a virtual environment to support a P2V migration is just the beginning. Managing the vCenter requires the ongoing collection of; you guessed it, performance metrics.

CIQ and Hyper9 are two examples of third party tools that can assist in the collection and analysis of vCenter data. If you organization is on a tight budget, vCenter data can be obtained using free PowerCli scripts. Attachment I provide the PowerCli scripts used to collect the data used to create Figure 14 and Figure 15. For a detailed explanation on PowerCli refer to [6].

Once KPIs are available, time series analysis trending or forecasting of virtual data center resource utilization can be accomplished. The "Host System" growth graph shown in Figure 14 was produced using Excel with data collected by running script 3. The trend graphs 6 months of "Host System" growth and trends that the number of "Host System" will double over the next 7 months. The data return by script 3 will also show which VMs are running on each "Host System".

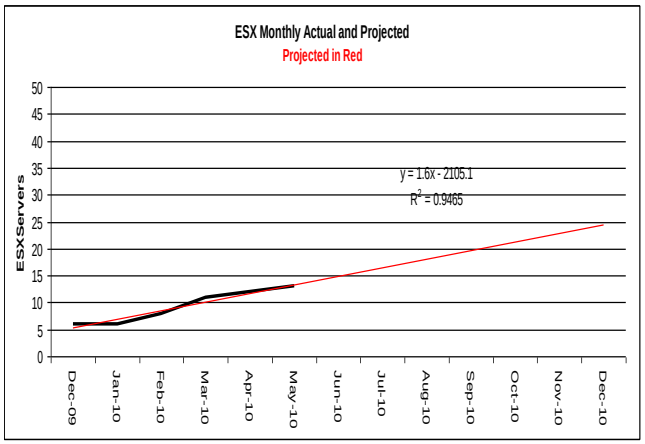


Figure 14: "Host System" Trending

Another important capacity management concern is over/under allocation of VM resources. Deploying over allocated VMs limits the number of VMs that can be deployed and can adversely affect scheduling performance and memory management. Under allocated VMs can affect end user experience. Neither is desirable. To produce Figure 15 a combination of script 3 and 4 were used. The chart shows weekly peak vCpu and memory usage for 5 VMs. VMs 1 and 2 are vCpu and memory over allocated and should be reconfigured.

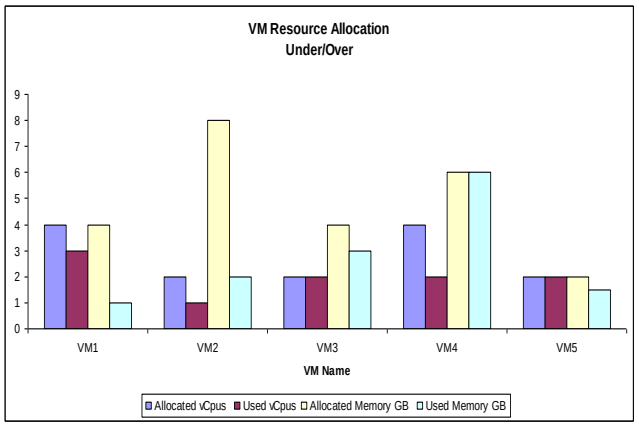


Figure 15: VM Under/Over

VI. Conclusions

The approach described in this paper has been used to plan the P2V migration of several critical applications. The initial "Host Systems" and VM configuration estimates were accurate; however tuning was required to improve response time for highly threaded applications. The problem with multi-threaded applications is not the availabilty MHz but thread concurrency per execution schedule. In a 4 CPU physical system running at 200, 4 threads could run simultaneously. In a 2 vCpu VM configuration, only 2 threads could execute per schedule. A good indication of the number of vCpu to assign is discussed in section 3 - subsection "Estimating VM Resources".

An operational virtual datacenter must be continually managed. Preventing VM sprawl and maximizing VM sizing and placement is a key to maintaining a low cost highly efficient virtual datacenter.

References

[1] T. Van Vleck (2006), The IBM 360/67 and CP/CMS [Online]. Available: http://www.multicians.org/thvv/360-67.html.

[2] P. Denning, and J. Buzen, The operational analysis of queuing network models. *Computer Surveys, 10*(3), 225 - 261 (1978).

[3] D. Sheetz, L. Song, A. Rikun (2008), predicting the Relative Performance of CPU, in proceedings of CMG 2008.

[4] Scott Lowe (2010), Mastering VMWare vSphere 4. SyBex.

[5] SpecOrg [online]. Available: http://www.spec.org/cpu2006/results/cpu2006.html

[6] Hal Rottenberg (2009), Mastering VMWare Infrastructure with Windows Powershell. Sapien.

Trademarks

Glossary

New Host System – physical server used to run virtual machines (VMs)

Packing Factor – the number of VMs that can/should be assigned to a "New Host System"
VM – Virtual Machine

VM/CMS - Virtual Machine/Conversational Monitor System

Minimal VM configuration – smallest resource allocation needed to provide acceptable performance

Attachment I

Open a connection

```
Connect-VIServer -Server sssssss -Protocol https -User uuuuuuuu -Password pppppppp
```

Close a connection

```
Disconnect-VIServer -Server sssssss -Confirm:$false
```

List all VMs by ESX by vCenter

```
$fileName = $Server + (get-date -format 'yyyyMM') + ".txt";
    get-vm | where-object -FilterScript {$_.PowerState -eq "PoweredOn"} | select host, name, NumCpu, MemoryMB |
    sort host, Name | ft -groupby host -auto | out-file -encoding "UTF8" -filepath $fileName -Confirm:$false
```

Show all VMs using less/more than a fixed CPU%

```
    #variables – past days, number of samples per VM, interval in minutes, CPU percentage
    #memory metrics can be obtained by changing cpu.usage.* to mem.usage.*
    $s            = "VMachine,MetricId,TimeStamp,Value"
    $fileNameOver  = "OverAllocated"  + (get-date -format 'yyyyMM') + ".csv"
    $fileNameUnder = "UnderAllocated" + (get-date -format 'yyyyMM') + ".csv"
    $s | out-file -filePath $fileNameOver  -append
    $s | out-file -filePath $fileNameUnder -append
    $VMs = get-vm | where-object -FilterScript {$_.PowerState -eq "PoweredOn" -and $_.NumCPU -ge 1}
    foreach ($vm in $VMs) {
        $Stats = get-stat -Entity $vm -STAT cpu.usage.* -Start (get-date).AddDays(-10) -MaxSamples 1 -
        IntervalMin 1
        forEach ($sample in $Stats) {
            $sample | forEach-object -process {
          $s = [string] $vm + "," + [string]($sample.MetricId) + "," +
              [string]($sample.Timestamp) + "," + [string]($sample.Value)
                    if ([int]($sample.Value) -ge 50 ) {
                        $s | out-file -filePath $fileNameUnder -append
                    } else {
                        $s | out-file -filePath $fileNameOver  -append
                    }
        }
    }
}
```

Engineering a University-Wide Event Calendar System

Jon Whitener and Kevin Daimi
Department of Mathematics, Computer Science and Software Engineering
University of Detroit Mercy
4001 McNichols Road, Detroit, MI 48221
{whitenjg, daimikj}@udmercy.edu

ABSTRACT

Online event calendars are powerful marketing tools for universities, enabling institutions to engage not only people on campus, but prospective students, alumni, and donors as well. Well-designed event calendars can be used to connect audiences with the activities that increase their bond with the university. In this paper, software engineering techniques and methods guide our design of a university-wide event calendar system (UWECS). Event information can be accessed and disseminated publicly by Internet services. UWECS will allow viewers to search for events on criteria, such as event categories, and request event reminders.

Keywords

Event Calendar, Software Engineering, Business Processes, Functional Requirements, Design

I. INTRODUCTION

Software engineering is an engineering discipline, which is concerned with applying agile and adaptable processes that leads to high-quality product meeting the needs of the stakeholders [19], [19] and [25]. Requirements engineering is the first key sub-process following the conclusion of a statement of needs. It has to do with the identification of goals to be accomplished by the potential system, the filtering out of such goals, and manipulating them to determine the specifications and constraints. Software requirements characterize what should be done to satisfy the needs of the stakeholders [13]. Software requirements must satisfy the stakeholders' needs to ensure the right system is later developed. The process by which these needs are identified is referred to as requirements engineering (RE).

One of the main objectives of RE is to enrich system modeling and analysis potentials so that organizations can better comprehend vital system aspects before they actually develop the system [22]. The functional requirements together with quality attributes and other nonfunctional requirements will establish the software requirements specification [27]. Functional requirements are the domain specific capabilities of the system. They represent what the developed system will do without any regards to how it should be done. However, nonfunctional requirements, or quality requirements, are constraints on the functional requirements. The non-functional characteristics of the business are believed to be harder to encapsulate in business process modeling, as the focus of the modeling is on functional behavior [17]. There are a number of techniques to modeling, representing, and checking requirements. Some of the approaches are: Case-Driven, Viewpoint-Based, Behavioral Pattern Analysis (BPA), Software Architecture Orientation, and Formal Methods approaches [1], [9], [14], [19], and [24].

Universities depend on revenue from student enrollment and financial donations (primarily from alumni). Income from these sources may be increased by showing that the institution is a lively, interesting place with many worthwhile events and activities. An obvious strategy for promoting that vibrant image is to publish information about the many events and activities occurring on campus. A natural choice for publishing such information is an event calendar. An event calendar made available on the Internet is an excellent way to make the information available to many people at a low cost. However, it is not enough for a university to have an online event calendar system. The system must be affordable, easy to use, and meet the needs that stakeholders have – needs that can be impressively deep and detailed at a complex organization like a university.

Many universities find it difficult to purchase expensive commercial calendar systems, which may not meet all their needs. So if "buy it" is not a strong option, "build it" may be. However, developing even a relatively straightforward system like an event calendar at a university involves many diverse stakeholders, and a number of requirements related to the variety of uses of the system and the decentralized nature of the organization. Building a system in-house is no better than buying it if the result does not solve the problem.

For decades, researchers have looked at the use and design of electronic calendar systems in the context of the business office, and more recently in the domestic setting.

According to Kincaid [11], in 1985, a complaint reported by electronic calendar users stated that "the terminal cannot be taken out of the office." With today's wireless, Web-enabled computing, that's less of a problem. But other shortcomings persist, including some listed in Kincaid's study, like rigid data-entry limitations and inadequate calendar views.

Payne [18] pointed out some interesting ways to enhance electronic calendar interfaces to make them as browsable as paper versions. This could be accomplished with techniques, such as showing more detail for more proximate appointments, and hiding time periods that lack appointments.

Palen [16] described how a groupware calendar system (primarily intended for scheduling meetings) can affect individuals' use of their own calendars, and how the system and the usages co-evolved. Tullio, et al [26] discussed augmenting such groupware systems with features like predicting event attendance and supporting interpersonal communication. Crabtree [5] looked at design issues in moving groupware calendar systems out of the office and into the domestic setting[5].

While there are areas of overlap, there are some key distinctions between the individual, office, groupware, or domestic calendars discussed in this previous research and a university event calendar system. It should be obvious that an event calendar is not intended to track the fine-grained schedules of individual people, or to coordinate their meetings.

At the University of Detroit Mercy (UDM), an event calendar was developed in-house in 2002, but it failed to meet some needs (for example, the need to place an event in multiple event categories) and had a number of persistent, difficult-to-fix defects. An open source event calendar product was adopted in 2010 to replace the previous system. This avoided the expense issue, but again resulted in a system that did not meet all the university's needs (e.g. the need to allow insertion of images or Web links into event descriptions), and had important defects as well.

This paper describes the design of a university-wide event calendar system guided by software engineering principles. The system overcomes key defects, pains, and bottlenecks in the previous systems mentioned above. The event calendar system presented below is not only intended to notify people of events, but also to market to a variety of audiences an overall image of the organization as a lively, interesting place. Moreover, the paper focuses on the application of software engineering techniques to the University-Wide Event Calendar System, or UWECS, especially the development of its requirements and its integration into an existing information technology context.

II. REQUIREMENTS ENGINEERING

The first major task for developing a software system is to understand what is needed: Who will use the system? What functions do the users desire? Besides users, what other stakeholders will be affected by the system, and which of their needs are involved?

A key first task in developing the UWECS system requirements was to identify the stakeholders and their goals. These are shown in Table I.

TABLE I
UWECS STAKEHOLDERS AND THEIR GOALS

Stakeholder Category	Goals
Event Approver	Examine every event before publication, to make sure it is complete, accurate, appropriate, and written in the proper style.
	Ensure that only approved events are published by the system.
Event Contributor	Add events to be published on the public Web sites, online calendar pages, RSS feeds, etc.
	Revise existing events in the system, in order to modify the event data.
	Remove existing events from the system.
Event Viewer	See what events are taking place at the University.
	Filter events to see only those of a specific kind or time period.
	Receive e-mail reminders for selected events.
Web Manager	Provide up-to-date, accurate event information on Web pages.

Individuals representing each of the stakeholder categories were contacted, and an effort was made to

understand each person's technical skill level and his/her roles and responsibilities relevant to the system. An excerpt of this stakeholder information is presented in Table II.

TABLE II
UWECS STAKEHOLDER ROLES & RESPONSIBILITIES

Title (Skill Level)	Roles / Responsibilities
Web Information Specialist (expert)	Edit and publish events submitted by others Enter many events Responsible for event information on UDM Web sites Responsible for Web site content and some site design Responsible for training others on using Web systems
Associate Vice President for Marketing & Public Affairs (nontechnical user)	Ultimately responsible for content and accuracy of Web sites published event information Provide event information to be included in system Has authority over editorial and style decisions
Associate Vice President for Information Technology Services (expert)	Has authority over IT hardware and software, and interaction of event system with other systems Control user access to University systems
Associate Director, Alumni Relations & Special Events (nontechnical user)	Source of information for important events
Assistant Registrar (nontechnical user)	Heavy contributor on current calendar, responsible for academic calendar and many events
Food service contractor (nontechnical user)	Has (unfortunately) been using current calendar as their work scheduling calendar Would submit some legitimate events as well

Needs and desires about the system were elicited from stakeholders via interviews and questionnaires. Our stakeholders were asked questions regarding publishing events at UDM. Samples of these questions include:

- What goals/needs do you have?
- What barriers have you encountered?
- What do you like/dislike about current/previous event calendar systems?
- Ideally, how would you like the new system to work?
- Do you have any questions for me?
- Who else should be asked these questions?

After eliciting stakeholders' comments, the task turns into moving from their free-form comments towards proper software requirements. One technique is to extract *need statements* from the stakeholder comments. A need statement records a single product attribute, function, or property required by one or more stakeholders [8].

TABLE III
ELICITED COMMENTS & CANDIDATE NEED STATEMENTS

Stake-holder Role(s)	Elicited Comments	Candidate Need Statement
V, W	Events must be checked for accuracy, and should be edited to meet UDM copy-writing style.	Viewers and Web managers need events checked for accuracy and proper style.
C, V	Need to be able to cut and paste text from a Word.	Contributors need to "paste" text from client OS clipboard.
C, W	Events should be able to appear across different Web sites. Currently, to get an event on two sites, it has to be entered twice, which makes for duplication of effort.	Contributors and Web managers need events to appear on multiple Web sites.
V, W	Ideally, within the TitanConnect portal, I'd like to have pages for Athletics that shows their calendar of events, SLO with their calendar of events, etc. and then at some higher level a consolidated calendar that includes events deemed appropriate for all.	Viewers and Web managers need filtered/sorted events to display within the TitanConnect portal on multiple calendars.
A, C, W	WYSIWYG entry: Text formatting, Images, Links Categories: Separate events into sections by category Be able to add categories later	A, C, and W need to include text formatting, Web links and images in event descriptions.
		A, C, and W need to use a WYSIWYG interface to enter event info.
		A, C, and W need to tag events with categories.
		A, C, and W need to add categories to the system as needed in an ongoing fashion.

After combining similar need statements, we have a needs list. The statements in the needs list were prioritized, so that the list could then serve as the basis of the UWECS software requirements specification (SRS). Table III shows a few collected stakeholder comments, alongside the candidate need statements extracted from those comments. Each candidate need statement was given a reference number (not shown), which were referred to in the SRS. Stakeholder roles are indicated with letter codes, as follows:

- A = Approver of events for publishing
- C = Contributor of event information
- V = Viewer of events and event calendar
- W = Web manager for Web site that does/ would include events

III. BUSINESS PROCESSES

To ensure that a software product has real value for an organization, it is important to explicitly identify the business processes that the product shall support. Wider in scope than a use case, a business process is a set of related tasks that are performed to achieve an intended business outcome, i.e. an outcome with value toward the organization's goals. Business processes usually have a

defined customer who receives the outcome, as well as a triggering event and a visible result.

A comprehensive set of an organization's business processes would provide a complete business model for the organization. In this paper, however, we limit our scope to those business processes related to the promotion of university events on an online calendar. For UWECS, the business processes are adapted from the "main user goals" in the SRS, and identified along with their triggering events, outcomes, and customers in Table IV. In UML notation, business processes can be depicted using use case symbols with the stereotype «workflow», as shown in Figure 1. According to Oestereich [15], "At this point the [business process] use cases are reduced to the intentions of the external commercial actors and described as abstractly as possible."

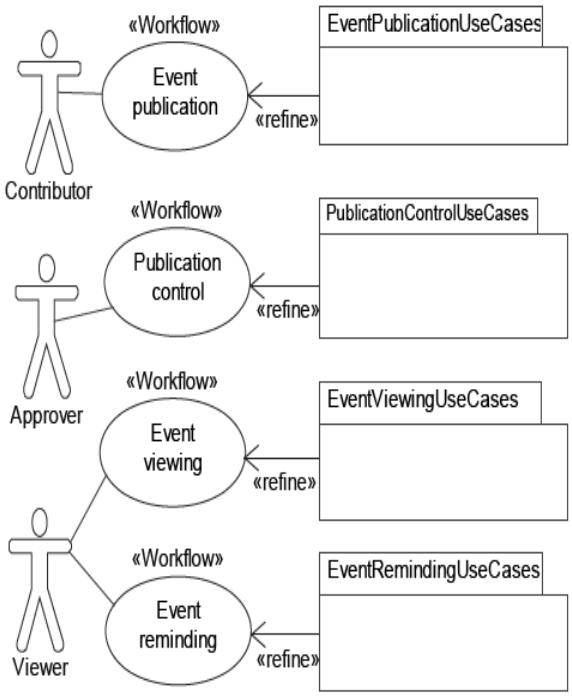


Figure 1: UWECS Business Process Diagram

Note that these business process workflows are given names that are noun phrases, unlike regular use cases, which use verb phrases. Also, since this is a high-level business view of the system, a business process model includes the *customers*, but not necessarily all of the system *actors* involved. A business process can be decomposed into a number of more-specific, conventional use cases, which should depict all system actors involved in each one.

TABLE IV
BUSINESS PROCESSES, TRIGGERS, OUTCOMES AND CUSTOMERS

Business process	Trigger	Outcome	Customer
Event publication	Contributor accesses system	Event appears online	Contributor
Publication control	Contributor submits event	Only appropriate events are published	Approver
Event viewing	Viewer requests event information	Viewer receives requested information	Viewer
Event reminding	Viewer requests event reminder	Viewer receives event reminder	Viewer

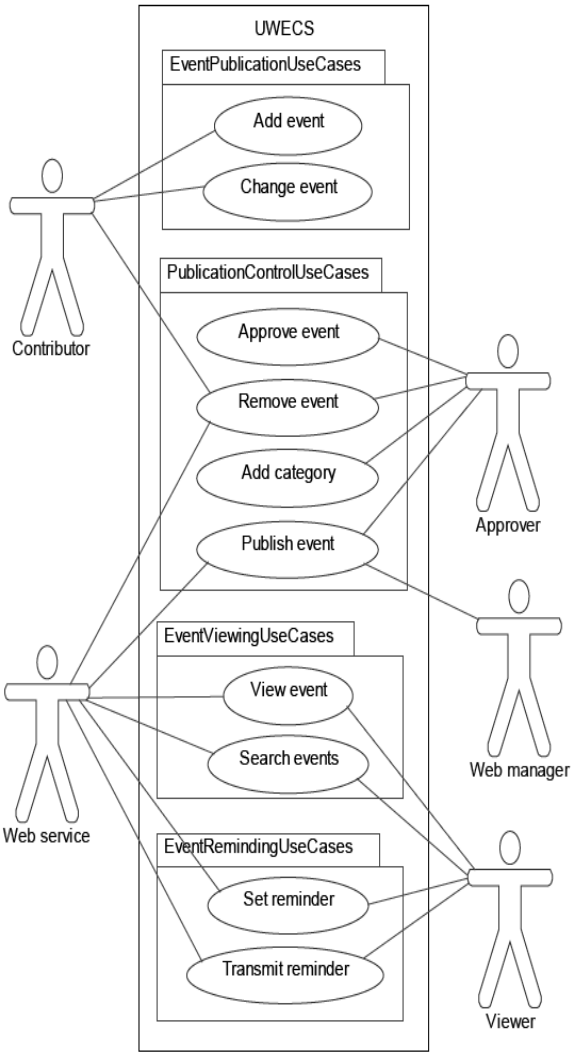


Figure 2: UWECS Use Case Diagram

IV. USE CASES

Once the business processes to be addressed by a software product are clearly understood, the next step is to break the processes down into more specific use cases. This decomposition of each business process is indicated in Figure 1 with UML package symbols. Each package represents a set of use cases and points to a business process with a «refine» relationship dependency arrow, indicating that the package of use cases is a refinement of the business process.

Booch et al. [3] stated that a use case "must be a *complete* flow of activity, from the *actor's point of view*, that provides *value* to the actor." Table V shows the UWECS decomposition of business processes into use cases, and the UML use case diagram is displayed in Figure 2. Use cases developed for the UWECS system are intended to be close to the direct user experience at the system level, though still implementation-independent.

TABLE V
Business Processes Decomposed into Use Cases

Business process	Use cases
Event publication	Add event Change event
Publication control	Approve event Remove event Publish event Add category
Event viewing	View event Search event
Event reminding	Set reminder Transmit reminder

In addition to the use case diagram, building the use case model involves the development of use case descriptions. For each use case, a use case description is written to describe the flow of activity for the use case. A use case description includes a basic flow, which tells the normal start-to-finish process for the use case. In addition, a use case description identifies extensions to the basic flow. Extensions are alternate courses of action that may occur during the use case. A common extension is when activity branches off from the basic flow due to an error condition. Use cases can also branch off to other use cases, indicated by including the underlined name of the other use case.

Use case descriptions also include the involved actors, stakeholders and their needs, pre- and post-conditions, and triggers. Figure 3 provides the UWECS use case description for the "Approve event" use case.

***Use Case**: Approve event*
Actors: Approver
Stakeholders and Needs:
Approver needs to ensure events are appropriate, complete, accurate, and in proper style.
Contributor needs to publish events for public viewing.
Web Manager needs to have appropriate, complete, accurate, and properly-written event information on Web service.
Viewer needs to view appropriate, complete, accurate events.
Preconditions: An unapproved event is stored in the system.
Post conditions: Approved events are appropriate, complete, accurate, in proper style.
Trigger: UWECS notifies an approver that an unapproved event awaits review.
Basic Flow:
1. UWECS notifies an approver that an unapproved event awaits review.
2. The approver checks the event for appropriateness, completeness, accuracy, and style.
3. The approver approves the event.
4. UWECS stores the event as an approved event.

Extensions:
The event requires changing which the approver can do.
The approver can Change event, and activity resumes.
The event requires changing which the approver cannot do.
The approver sends the event back to the contributor with a note describing how the contributor must Change event, and the use case ends.
The event is unacceptable.
The approver Rejects the event, optionally adding an explanatory note, and the use case ends.

Figure 3: Approve Event Use case

V. UWECS FEATURES

The key general functions that the UWECS is designed to provide are discussed below.

A. *Functionality*

The overall purpose of UWECS is to enable users to promote their university events online. UWECS accepts as input information about university events, including event date and time. UWECS stores the information persistently, and allows modifications to the information. The output of UWECS is the event information, made accessible to external systems.

Users enter their event information into UWECS, so that various Internet services can access the information and disseminate it publicly. Internet services include Web sites and other online services, such as RSS feeds. UWECS allows viewers to search for events on criteria such as text-matching, date range, and event categories. Event submission is controlled and secure. Also, UWECS enables the university's Marketing & Public Affairs Department to review and approve event

information before events appear publicly. Furthermore, the system enables viewers to request e-mail reminders for selected events.

A key benefit of UWECS (and one that differentiates it from its predecessors) is the ability of users to tag events with multiple category values, i.e. descriptive labels that help people find events of interest to them. Category values can indicate:

- Audiences (e.g. alumni, students, faculty)
- University units (e.g. engineering, library, athletics)
- Types of event (e.g. presentation, game, exhibit, volunteer opportunity)
- Sponsors and/or involved organizations
- Themes (e.g. diversity, social justice)

Users and systems can filter events according to category tags in order to display only those events of particular relevance, for example, only events tagged "alumni" for a page on an alumni Web site. Certain privileged UWECS users can add new category values to the system at any time.

B. Data

The fundamental unit of information in UWECS is the event. Event data fields include:

- Event title, summary, and full description
- Event date
- Begin time, end time (or "all day")
- On-campus or off-campus location, building/area, room
- URLs of related Web pages, if any
- Admission fee
- Invitation status (by invitation only, students only, faculty only, staff only, alumni only, or some combination; or open to the public)
- RSVP status (unnecessary, requested, required)
- Publish date (for deferred publication)
- Preview image
- Sponsor(s)
- Contact name, phone, e-mail

The event category, discussed above, is another data element demanding consideration. Looking at two types of category tags may give a better idea how they help users filter events. Event categories could be used to indicate intended university audiences for an event, and, orthogonally, the relevant schools. Audience category values could include faculty, staff/administrators, current students, alumni, prospective students, parents, and the public. School category values could include Architecture, Business, Engineering, Law, and Liberal Arts.

A good example of the need to specify multiple, orthogonal event categories is when the CEO of Ford Motor Company gives a presentation on campus. This event is of interest to both the Engineering school and the Business school. Depending on who is invited, many UDM alumni would be interested (the D is for Detroit, after all), as would some faculty and students. All of these categories make a meaningful connection between the purpose of the event and the relevant schools and audiences.

VI. UWECS FUNCTIONAL REQUIREMENTS

Functional requirements for a software system must state clearly what the system must do, with enough detail to allow system design and testing. A comprehensive list of such requirements forms the basis for system development, and is typically contained in a software requirements specification document, or SRS. Any non-trivial system will have a lot of functional requirements. Within an SRS, there are several ways to organize them. The key criterion for organizing requirements is to make the SRS as understandable as possible to those who will read it.

For UWECS, the functional requirements were organized into four groups, each containing requirements related to one of the main user goals: *publish event*, *control event publication*, *view event information*, and *receive reminders about event*. Additionally, each of the four groups also included related external interface requirements, in order to place requirements where they made most sense. Below is a sample of some of these UWECS requirements. Requirement identification numbers are omitted; numbers in parentheses refer to the relevant need statements (Table 3) that were developed during requirements elicitation.

A. Publish Event

1) External interface requirements

- The system must provide a user interface for contributors to enter new event information. (11, 16)
- The system must provide a user interface for contributors to modify existing event information. (6, 42, 60)

- The system must provide a user interface for approvers to publish events. (2)

2) Functional requirements

- The system must enable contributors to submit event information. (11, 16)
- The system must enable contributors to edit the information in previously-submitted events. (6, 42, 60)
- The system must enable contributors to include text formatting – bold and italic text; bullet lists; headings – in event information. (32)
- The system must enable contributors to include Web links in event information. (14, 32, 63, 64)
- The system must enable contributors to include images in event information. (21, 32)
- The system must allow contributors to tag events with categories. (24, 34, 50)
- The system must publish events to multiple targets, including public Web sites, the UDM TitanConnect portal, and an RSS feed. (2, 15, 17, 22, 28, 48, 62)

B. Control Event Publication

1) External interface requirements

- The system must provide a user interface for approvers to review event information. (3, 6, 18, 42, 60, 61)
- The system must provide a user interface for contributors to remove existing event information.
- The system must provide a user interface for approvers to add event categories.

2) Functional requirements

- The system must require approval by an approver for each event before publishing. (3, 6, 18, 42, 60, 61)
- The system must enable approvers to add new categories, as needed, in an ongoing fashion (so that new categories will become available event tags). (35, 52)
- The system must restrict event publication to approved events only. (61)
- The system should control event content as much as possible through use of structured data forms. (43, 55)

C. View Event Information and Receive Reminders

1) Functional requirements

- The system must provide ways for viewers to browse events and navigate event calendars, including monthly calendar grids (i.e. a set of links arranged in a table, like a printed monthly calendar). (12, 29, 37, 41)
- The system must enable Web managers and viewers to display only those events that match specified event category tags or combination of tags. (25, 30, 51, 53)
- The system must enable viewers to view event information that includes text formatting. (32)
- The system must enable viewers to view images in event information. (21, 32)
- The system must enable viewers to search for events based on specific event category tags or combination of tags. (25, 30, 51)
- The system must enable Web managers and viewers to display only events that match specified event date/time ranges, including by month. (10, 56)
- The system must provide viewers and contributors access to past events. (13, 27, 31)
- The system must support access to event information via persistent URLs. (64)
- The system should support simple inclusion of event information into Web pages. (14, 47, 48)
- The system should enable events and calendars to appear in the TitanConnect intranet portal Web site. (20, 62)
- The system should provide filtered/sorted events for display within the TitanConnect portal on multiple calendars. (20)
- The system must enable event viewers to request e-mail reminders for specific events. (59)

VII. SHIFTING FROM PRODUCT DESIGN TO ENGINEERING DESIGN

Fox [8] differentiates between software product design and software engineering design. He defines software product design as "the activity of specifying software product features, capabilities, and interfaces to satisfy client needs and desires," contrasted with software engineering design, defined as "the activity of specifying programs and sub-systems, and their constituent parts and workings, to meet software product specifications."

In simpler terms, software product design defines what the product should do, and software engineering design determines how the product should be built. For the UWECS system, the product design process consists most importantly of the development of the software requirements specification (SRS) and the use case model. These are both common software engineering techniques that define what the product should do. With the SRS and use case model in hand, attention turns to engineering design: how to build a system that provides the features and functions specified by the product design?

As we approach engineering design, it should be noted that this paper focuses on the development of UWECS as a software system that is dependent on, and integrated with, external systems that already exist as parts of the system's information technology (IT) context. This consideration is especially important due to a number of constraints on the UWECS system, contained in the SRS:

- The UWECS system must be implemented on hardware, software, and network platforms already in place in the University of Detroit Mercy IT environment.
- There is no budget for any additions to these hardware or software platform elements for UWECS.
- The deadline is strictly short.
- There are limited human resources.

In addition to the project constraints, there is a goal related to usability. During the requirements elicitation process with stakeholders, it became clear that users dislike having to learn how to use new, unfamiliar systems. This relates directly to the effort needed to learn UWECS, and ultimately to its successful adoption across the organization. If the system's users decide that it takes too much work to learn, the system is at risk of being ignored by them. This will destroy the intent of the whole project.

The best way to avoid this problem is to provide user interfaces that users are already familiar with. For UWECS, an existing Web content management system (WCMS) can serve as the primary interface for submitting, editing, and approving events. Users already use this WCMS to maintain their various Web sites, so they would need very little training on how to use it for the UWECS functions. This means, of course, that the integration of the WCMS as part of UWECS must be carefully engineered. The features and constraints of the WCMS must be dealt with in detail. In this case, an existing "external" system in the current IT infrastructure becomes part of the "internals" of the system under development. This is only one example of how UWECS integrates with the existing software and hardware context. Other examples include:

- UWECS must provide event information to the university's TitanConnect intranet Web portal.
- UWECS must provide event information, and event searching capability, to several public university Web sites.
- The reminder function of UWECS requires the sending of e-mail messages. For security and abuse reasons, e-mail delivery is very tightly-controlled by university IT policies. No additional mail servers will be allowed, so UWECS must work in conjunction with the existing mail server system, and within that system's rules and restrictions.

So, as focus shifts from product design (what should the product do) to engineering design (how should the product be built), much attention is paid to how UWECS interacts with the existing IT infrastructure.

VIII. UWECS DESIGN

A. System Architecture

As discussed above, a key design concern for the UWECS system is its integration into the existing IT infrastructure, to the point of using existing components as parts of UWECS. Figure 4 exhibits a UML deployment diagram, which shows the main interactions between the important nodes. Network protocols are indicated on the communication path lines between the nodes.

The two most interconnected nodes in the diagram are the WCMS and the Web Server, which are the two primary components of UWECS. As mentioned above, the WCMS is the existing Web content management server, which is already in use by many stakeholders for managing the content of their Web sites. The WCMS provides tested and familiar interfaces for creating, modifying, approving, and removing content intended for Web delivery. The WCMS provides a word processor-like interface for editing content, which allows for familiar text styling methods and the inclusion of images and Web links.

Contributor and Approver connect to the WCMS through a Web-based interface over HTTPS. Contributors create and modify event information within the WCMS, and upon submission, the WCMS

inserts the submitted event into an approval workflow. Approvers must then approve the content before it is published.

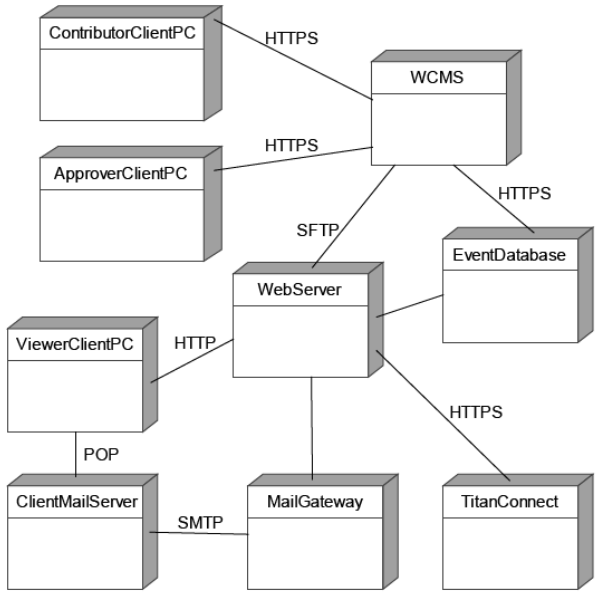


Figure 4: UML Deployment Diagram

The WCMS is able to publish the event information in multiple formats to multiple destinations. A viewable HTML version of each event is published to the Web Server as a conventional Web page. In addition, the WCMS publishes the structured event data to the Event Database, which can be queried by scripts on the Web Server node for calendars, event searches, category-specific listings, etc.

The Web Server shown in the deployment diagram represents the main university Web server, which, in addition to serving Web client requests, feeds the event information to the TitanConnect intranet portal. Other, separate university Web server nodes (not shown in the diagram) are also able to query the Event Database, but would not feed TitanConnect.

The deployment diagram shows the nodes involved when a Viewer accesses the Web site to view event information, as well as when requesting an e-mail reminder for an event. The UWECS component that is responsible for handling reminders resides on the Web server node. Once a day, the reminder component queries the Event Database – a "Reminders" table – to discern who should receive one that day. A reminder message is assembled and sent to the university's Mail Gateway service, which in turn delivers an e-mail message to the recipient Client Mail Server. Ultimately, the Viewer Client PC receives the reminder e-mail.

B. Interface Design

Most UWECS users will interact with the system either through the WCMS or through Web pages delivered by a Web server. Contributors and Approvers will use the existing WCMS interface. The familiarity of the interface is an important advantage of the design, since it will require little training for those already using it. Figure 5 illustrates an example of the WCMS interface. From the Viewer's point of view – typically seeing an event listing on a Web site – there are a number of interface formats. Figures 6 and 7 reveal just two of the different ways that Web sites present event information.

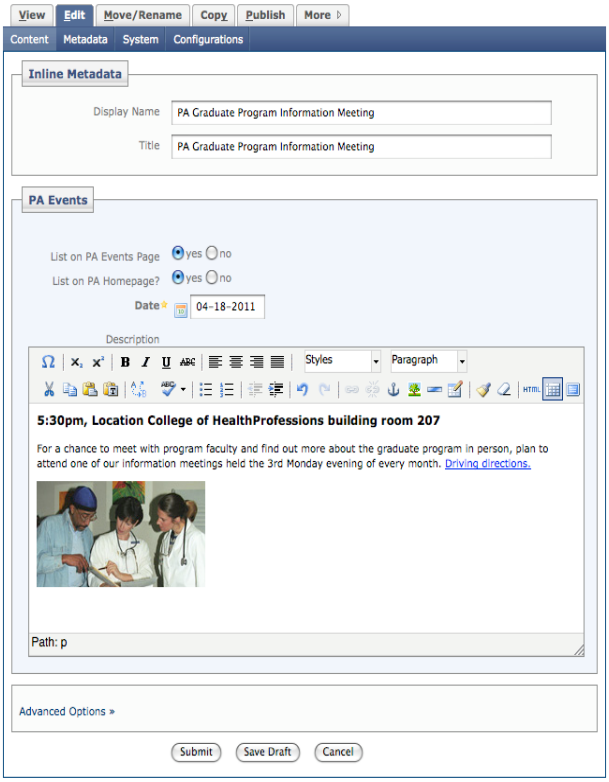


Figure 5: WCMS Interface

UWECS must therefore provide event information in a manner that can be easily transformed into various layouts and designs. A key concern, then, is to keep any presented information separate from the event data itself. This recalls the software engineering pattern often referred to with Web applications, Model-View-Controller or MVC [12]. A key concept of the MVC pattern is that aspects of the view (e.g. design, formatting, and layout) are kept independent of the data, which are part of the domain model.

IX. Non-Functional Requirements

In addition to the functional requirements, which indicate what the system should do, it is important to clearly understand some of the non-functional qualities that the system must have. A few are listed here. Again, the numbers in parentheses refer to stakeholder need statements. This provides a level of traceability so that we can assure that requirements are actually related to known needs or constraints.

- The system should support a minimum of 8,000 event calendar views per day on UDM Web sites. (23)
- System users as described above should be able to use the system reliably with a maximum of 30 minutes training. (7, 46)
- Browsing events on the Web, including navigation between event and calendar views, should be easy to use for the Web users with average to slightly below average Web browsing skills. (12, 16, 29)
- Writing code to insert event information into a Web page should require less than 30 minutes. (47)
- The system must prohibit unauthorized access to the event database.

X. Conclusions

An online event calendar system is the ideal approach to promote happenings at a university. After experiencing a couple of failures in the implementation of an online event calendar system for University of Detroit Mercy, software engineering practices were applied for the first time to the development of a new system to better meet users' needs.

Stakeholders were consulted to discover their particular and various desires, and their needs were systematically converted into system requirements. Also, the components of the system's hardware and software environment were integrated as parts of the design in order to meet a number of non-functional requirements. For the first time, users have an online event calendar system that is based directly on their needs, allowing Contributors to submit events, Approvers to control events, and enabling Viewers to both view events in multiple formats, and receive event reminders, all without the need to learn new system interfaces.

In addition to the advantages that software engineering provides in system development, the documentation developed through the software engineering processes will mean that future modifications to the system can be based on a solid understanding of development decisions and rationale.

References

[1] A. El-Ansary, "Behavioral Pattern Analysis: Towards a New Representation of Systems Requirements Based on Actions and Events," *in Proc. ACM Symposium on Applied Computing (SAC)*, Madrid, Spain, 2002, pp.984-991.

[2] B. W. Boehm, "Software engineering," *IEEE Trans. on Computers*, vol. 25, no. 12, pp. 1226-1241, Dec. 1976.

[3] G. Booch, R. A. Maksimchuk, M. W. Engel, B. J. Young, J. Conallen, and K. A. Houston, *Object-Oriented Analysis and Design with Applications*, Boston, MA: Addison Wesley Professional, 2007, pp.177-177.

[4] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, New Jersey: Addison-Wesley, 2005, pp. 249-249.

[5] A. Crabtree, T. Hemmings, T. Rodden, and J. Mariani, "Informing the Development of Calendar Systems for Domestic Use," *in Proc. European Conf. on Computer Supported Cooperative Work (ECSCW'03)*, 2003, pp. 119-138.

[6] B. Desruisseaux, et al. (2009, September) *iCalendar Transport-Independent Interoperability Protocol (iTIP): Scheduling Events, Busy Time, To-dos, and Journal Entries, RFC 2446* [Online]. Available: http://www.ietf.org/rfc/rfc2446.txt

[7] L. Dusseault and J. Whitehead, "Open Calendar Sharing and *Scheduling* with CalDAV," *IEEE Internet Computing*, vol. 9, no. 2, 81-89, Mar. 2005.

[8] C. Fox, *Introduction to Software Engineering Design: Processes, Principles, and Patterns with UML2*, Boston: Addison Wesley, 2007.

[9] J. Hausmann, R. Heckel, and G. Taentzer, "Detection of Conflicting Functional Requirements in a Use Case-Driven Approach," *in Proc. International Conference on Software Engineering (ICSE'02)*, Orlando, Florida, USA, 2002, pp. 105-115.

[10] IEEE Recommended Practice for Software Requirements Specifications, IEEE Std 830-1998.

[11] C. M. Kincaid and P. B. Dupont, "Electronic Calendars in the Office: An Assessment of User Needs and Current Technology," *ACM Transactions on Office Automation Systems*, vol. 3, no. 1, pp. 89-102, Jan. 1985.

[12] G. E. Krasner and S. T. Pope, "A Cookbook for Using the Model-View Controller User Interface Paradigm in Smalltalk-80," *J. Object Oriented Program*, vol. 1, no. 3, pp. 26-49, Aug. 1988.

[13] A. van Lamsweerde, *Requirements Engineering: from System Goals to UML Models to Software Specification*, Wiley, 2009.

[14] H. Mei, "A Complementary Approach to Requirements Engineering – Software Architecture Orientation," *Software Engineering Notes*, vol. 25, no. 2, pp. 40-47, 2000.

[15] B. Oestereich, *Developing Software with UML: Object-Oriented Analysis and Design in Practice*, 2nd ed. London, U.K.: Addison Wesley Professional, 2002.

[16] L. Palen, "Social, individual & technological issues for groupware calendar systems," *in Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '99)*, 1999, pp 17-24.

[17] C. Pavlovski, and J. Zou, "Non-Functional Requirements in Business Process Modeling," *in Proc. the 5th Asia-Pacific Conference on Conceptual Modeling (APCCM 2008)*, Wollongong, Australia, 2008, pp. 103-112.

[18] S. J. Payne, "Understanding Calendar Use," *Human-Computer Interaction*, vol. 8, pp. 83-100, 1993.

[19] S. L. Pfleeger, and J. M. Atlee, *Software Engineering Theory and Practice*. New York: Prentice Hall, 2010.

[20] J. van der Poll, and P. Kotzé, "Combining UCMs and Formal Methods for Representing and Checking the Validity of Scenarios as User Requirements," *in Proc. the Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT)*, 2003, pp. 59-68.

[21] R. Pressman, *Software Engineering: A Practitioner's Approach*, McGraw Hill, 2005.

[22] W. Robinson, S. Pawlowski, and V. Volkov, "Requirements Interaction Management," *ACM Computing Surveys*, vol. 35, no. 2, pp. 132-190, 2003.

[23] P. Sawyer, "Software requirements," in *Software Engineering*, 3rd ed. R. H. Thayer and M. J. Christensen, Eds. New Jersey: Wiley, 2005, pp. 125-139.

[24] A. Silva, "Requirements, Domain, and Specification: A Viewpoint-Based Approach to Requirements Engineering," *in Proc. International Conference on Software Engineering (ICSE'02)*, Orlando, Florida, USA, 2002, pp. 94-104.

[25] I. Sommerville, *Software Engineering.* New York: Addison Wesley, 2010.

[26] J. Tullio, J. Goecks, E. D. Mynatt, and D. Nguyen, "Augmenting Shared Personal Calendars," *in Proc. Annual ACM Symposium on User Interface Software and Technology (UIST '02)*, 2002, pp. 11-20.

[27] K. E. Weigers, "Karl Wiegers Describes 10 Requirements Traps to Avoid," *Software Testing and Quality Engineering*, vol. 2, no. 1, pp. 34-40, 2000.

Table-based Software Designs: Bounded Model Checking and Counterexample Tracking

Noriyuki Katahira[1], **Weiqiang Kong**[1], **Wanpeng Qian**[1],
Masahiko Watanabe[2], **Tetsuro Katayama**[3], **Akira Fukuda**[4]
[1]Fukuoka Industry, Science & Technology Foundation, Japan, {*katahira, kong, qian*}*@lab-ist.jp*
[2]CATS Co., Ltd., Japan, *watanabe@zipc.com*
[3]Faculty of Engineering, University of Miyazaki, Japan, *kat@cs.miyazaki-u.ac.jp*
[4]Graduate School and Faculty of ISEE, Kyushu University, Japan, *fukuda@ait.kyushu-u.ac.jp*

Abstract— *Model description languages used by most software model checkers are typically program-like languages such as the Promela language for the well-known model checker Spin. To promote practical use of model checking techniques in on-site software development, we realized, however, that graphicalized modeling languages (e.g., representatively, UML) are more easily acceptable compared to model-checker-specific program-like formal description languages. In this paper, we propose a model checking technique for software designs developed in a table-based description language – State Transition Matrix (STM), which is commonly considered to be effective for embedded software development. In addition, we also describe our proposed approach for graphically tracking counterexamples (i.e., design errors w.r.t. specified properties) reported by model checking. Supporting both graphicalized model description and graphically counterexample tracking is believed by us to be important for enhancing usability of model checking techniques and tools for on-site software engineers.*

Keywords: State Transition Matrix, Bounded Model Checking, SMT Solving, Counterexample Tracking

1. Introduction

State Transition Matrix (STM) is a table-based model description language. Although primarily used for software development, STM is capable to describe general automaton-based system behaviors, e.g., human's operation etc. As a basic rule, the horizontal axis of a STM table declares *status* that a system under consideration could be in, the vertical axis declares *events* that may be dispatched to the system, and a row-column intersection *cell* declares behaviors (actions and transition target status) of the system when a specified event is dispatched in a specified status. A whole system could be described by defining each of its sub-systems as a STM and the sub-system STMs execute in an interleaving manner and communicate through shared variables or asynchronous message passing.

STM has been widely accepted and used in software industry, and thus been adopted as the model description language of some commercial model-based CASE tools such as ZIPC [1]. However, although its wide acceptance and usage, there is lack of formal verification supports for conducting rigorous analysis to improve reliability of STM designs, e.g., ZIPC provides facilities for syntactic check and test only. Therefore, we started to implement a model checking tool called *Garakabu2*, which could be used to formally analyze correctness of STM designs (typically consisting of multiple STMs) developed using ZIPC w.r.t. user-specified Linear Temporal Logic (LTL) properties [2].

There are quite a few model checking tools, e.g., *Spin* [3] and *CBMC* [4], available for both academic and practical industry usage. In this paper, we focus only on the latter usage, which is also the primary target of Garakabu2. We have observed through our investigation in Japanese software industry that formal verification (primarily model checking) techniques are mostly applied, so far, after the development of source codes. To employ Spin-like model checkers, engineers have to re-describe the (parts of the) system (which could be a design, source codes, or others) to be checked with a model-checker-specific description language, in the Spin case, *Promela*. This re-description process is error-prone and thus it is non-trivial to guarantee the consistency between the original system and the re-described one. CBMC-like model checkers could be used directly to analyze correctness of source codes, in the CBMC case, ANSI C and C++ source codes. However, our argument is that quite a lot source code errors are due to design errors, and therefore, detecting errors in an early design phase of the software development process could greatly reduce development time and costs.

Garakabu2 is built to detect software design errors. Designs developed using STM by engineers for on-site software development could be checked by Garakabu2 just as they are, and thus engineers do not need to learn a "second" model-checker specific description language for conducting model checking. Another feature of Garakabu2 is that the execution sequences of counterexamples (i.e., design errors) reported could be tracked graphically (in the form of table) in STMs in the ZIPC environment. Such *visible* execution is expected to be useful for engineers to easily understand the reasons for design errors. We believe that these two features

☐ CHANGER		IDLE	WAIT_REQUEST		WAIT_MONEY_TAKEN
		0	1		2
xChangePrepare	0	WAIT_REQUEST balance=balance+10000;	/		/

x10KYenRequest	1	/	balance >= 10000
			WAIT_MONEY_TAKEN
			payment=10000;
			balance=balance-payment;
			paid=true;
				else
				IDLE
				payment=0;
				paid=true;
					/

taken	2	/	×		WAIT_REQUEST
					payment=0;
					taken=false;

☐ RETURNER		WAIT	RETURN	
		0	1	
paid	0	RETURN paid=false;	×	

xReceive	1		payment =/= 0
			WAIT
			taken=true;
				else
				WAIT
				taken=true;

Fig. 1: A Simplified Money-Changer System (MCS) consisting of a CHANGER and a RETURNER.

are effective to some extent for enhancing usability of model checking techniques and tools, and thus are helpful for promoting their practical use in on-site software development.

The rest of the paper is organized as follows. Section 2 introduces the table-based language STM. Section 3 describes the key techniques for Bounded Model Checking [5] STM designs. Section 4 discusses in detail our attempts made in Garakabu2 for enhancing its usability. Section 5 mentions future work and concludes the paper.

2. State Transition Matrix (STM)

We informally introduce the table-based language STM that uses shared-variable as the means of communication, whereas interested readers are referred to [6] for its formal definitions. A simplified Money-Changer System (MCS) shown in Figure 1 is used as our demonstration example.

MCS consists of two components modeled as two STMs. (1) A device called CHANGER, which supplies smaller denominations when equivalent amount of money in a larger denomination is inserted. The small denominations are delivered to another device called RETURNER. (2) Device RETURNER receives small denominations from CHANGER and waits until they are taken. MCS is modeled in a greatly simplified means by abstracting away details irrelevant to the demonstration purpose. Additionally, we intendedly introduced design errors to the system for demonstration purpose as well, e.g., MCS's behaviors are *unreasonably* defined when there are not enough small denominations, which will be discussed in detail in Section 4.

Taking STM CHANGER as an example. CHANGER could be in three *status*, i.e., `IDLE`, `WAIT_REQUEST`, and `WAIT_MONEY_TAKEN` with obvious meaning. Initially, all STMs are in their left-most status. Three *events* that are possibly dispatched to CHANGER include `xChangePrepare`, `x10YenRequest`, and `taken`. Events whose names are prefixed with "x" are called external events, and those without are called internal ones. An external event is dispatched by the environment where the system concerned resides in, and an internal event is dispatched by the execution of the system. External event `xChangePrepare` denotes initialization of CHANGER, external event `x10YenRequest` denotes a request to exchange a banknote of 10K denomination, and internal event `taken` denotes that banknotes of small denominations have been taken.

An event-status intersection *cell* declares the system's behavior when the specified event is dispatched in the specified status. There are three kinds of cells in a STM:

- *Normal cell*. A normal cell declares *actions* and *transition target status* of a STM. For example, the intersection cell `xChangePrepare-IDLE` declares that if event `xChangePrepare` is dispatched when CHANGER is in status `IDLE`, CHANGER's balance is increased by 10000 and after that it changes to status `WAIT_REQUEST`. A cell could be divided into sub-cells if *conditions* are defined to further restrict STM's behavior, as of the case `x10KYenRequest-WAIT_REQUEST` in which condition `balance`>=10000 is defined.
- *Ignore cell*. An ignore cell is denoted by a "/" in a STM. Meaning of an ignore cell is that the dispatch of an event in a status is ignored and nothing changes. For example, the intersection cell `x10KYenRequest-IDLE` means intuitively that a request to exchange when CHANGER is `IDLE` will just be ignored.
- *Invalid cell*. An invalid cell is denoted by a "×" in a STM. Meaning of an invalid cell is that an event should never be dispatched in a specified status. For example, the intersection cell `taken-WAIT_REQUEST` means intuitively that the event small denominations have been taken should not be dispatched when CHANGER is waiting for a request.

Regarding normal cells, instead of actions and transition target status, the name of another STM (named here as *called STM*) could be written in them to express that the concrete behaviors are as those defined in the *called STM*. STMs defined in this way is called *hierarchical* STMs. In this paper, we will not further discuss such kind of STMs although model checking them are supported by Garakabu2.

3. Bounded Model Checking (BMC) of STM designs with Garakabu2

The original model checking algorithms implemented in Garakabu2, as been described in [7], are *explicit-based* ones similar to those of SPIN. Basic idea of explicit-based algorithms is to exhaustively explore through explicit representation all reachable system states (starting from the initial one) [2]. However, concurrent software systems like STM designs generally contain tremendous number of possible interleavings of events and combinations of data values [8],

which usually blows up, if represented explicitly, the state space to be analyzed. Such a problem is commonly called the *state-explosion problem*. Satisfiability Modulo Theories (SMT) [9] based model checking techniques could attack this problem by symbolically representing and enumerating states. In this section, we describe the key techniques implemented recently in Garakabu2 for SMT-based Bounded Model Checking (BMC) of STM designs. We refer interested readers to [6] for a more formal (mathematical) description.

3.1 Encoding of STM Designs

Generally speaking, SMT is a technique to determine assignments of all the variables of a given logical formula to make the formula *true*, or no such assignment exists. The variables can be of types (such as integer and real) that have associated theories (such as the theories of linear integer arithmetic and linear real arithmetic, respectively), in which fixed interpretations are also given to non-logical operation symbols and functions.

Key idea of employing SMT techniques to conduct BMC of STM designs is to encode all execution sequences within a given bound (execution step) of a STM design, together with the *negation* of a LTL property to be checked, into a quantifier-free logical formula. Satisfiability of the formula is then determined by SMT solvers. If satisfied, a model of the formula (i.e., interpretation/assignments to all the state variables that make the formula *true*) is a witness of some bad behaviors (i.e., design errors) of the STM design that violate the LTL property. Since only bounded behaviors of the design are analyzed, this approach is primarily used for revealing design errors rather than proving their absence.

Figure 2 shows the overall process of BMC of STM designs with Garakabu2. A STM design could be viewed as a transition system whose behaviors are captured by an *initial system state* and a set of *transitions* that changes system states. A system state (hereinafter called *state* for simplicity) is a set of (all) state variables (hereinafter called *variables* for simplicity) declared and used in the STM design. For the MCS shown in Figure 1, a state is composed of:

{`xChangePrepare`:Bool, `x10KYenRequest`:Bool, `taken`:Bool, `balance`:Int, `payment`:Int, `paid`:Bool, `xReceive`:Bool, `chgStatus`:Int, `retStatus`:Int}

where "Bool" or "Int" written after each variable denote the type of the variable. Note that variables `chgStatus` and `retStatus` are additional variables introduced by Garakabu2 for representing respectively the current active status of each STM, where initially both of them have the value 0. Initial values of other variables are defined in a separate file in ZIPC which are omitted in this paper.

Encoding rule for the initial state is an *and*-conjunction of equations, each of which link a variable with its initial value. The initial state of MCS is encoded into the following formula, called *initFL*:

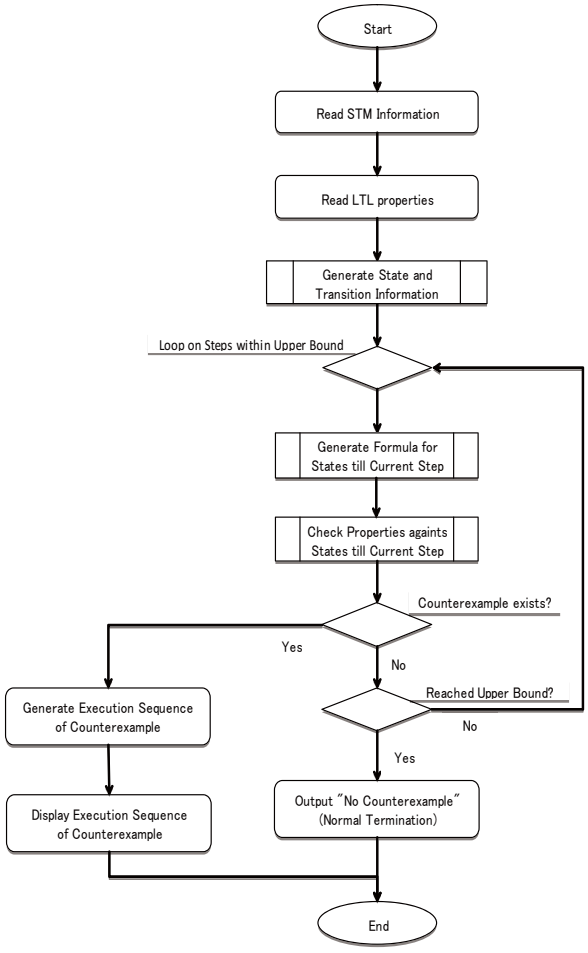


Fig. 2: Bounded Model Checking Process of Garakabu2

$\texttt{xChangePrepare\_0} = \textit{false} \wedge \texttt{x10KYenRequest\_0} = \textit{false} \wedge \texttt{taken\_0} = \textit{false} \wedge \texttt{balance\_0} = 0 \wedge \texttt{payment\_0} = 0 \wedge \texttt{paid\_0} = \textit{false} \wedge \texttt{xReceive\_0} = \textit{false} \wedge \texttt{chgStatus\_0} = 0 \wedge \texttt{retStatus\_0} = 0$

Note however that, each variable of a state is renamed by attaching to its name a suffixed index number. The reason behind is that a set of fresh/new variables is introduced for each execution step (including the step for initial state), and is used to uniquely characterize a state (or, symbolically, a set of states) when the system reaches that step.

Encoding rule for each execution step within a given bound is an *or*-conjunction of transitions that might possibly be executed from a previous state. There are two kinds of transitions for a STM design, one corresponding to the execution of a normal cell and another corresponding to the dispatch of an external event. For the MCS shown in Figure 1, there are 7 normal (sub-)cells (named as *c1* to *c7*) and 3 external events (named as *e1* to *e3*). We show the encoded formula, called *c1FL*, for normal cell *c1* – `xChangePrepare-IDLE` – in step 1 in the following, and the remaining cells could be encoded similarly.

$$\texttt{xChangePrepare\_0} = \textit{true} \wedge \texttt{chgStatus\_0} = 0 \wedge$$
$$\texttt{balance\_1} = \texttt{balance\_0} + 10000 \wedge \texttt{chgStatus\_1} = 1 \wedge$$
$$\texttt{xChangePrepare\_1} = \texttt{xChangePrepare\_0} \wedge$$
$$\texttt{x10KYenRequest\_1} = \texttt{x10KYenRequest\_0} \wedge \ldots\ldots \wedge$$
$$\texttt{retStatus\_1} = \texttt{retStatus\_0}$$

The two equations in the first line of *c1FL* characterize the *execution condition*, whereas equations in the remaining lines characterize the *execution effects*, of *c1* in step 1. The condition restricts that in the previous state, i.e. the state whose variables are with index number 0, event `xChangePrepare` is dispatched and CHANGER is in status 0. The effects express that the values of `balance` and `chgStatus` are changed and all the other variables remain unchanged (where some equations for unchanged variables are omitted in *c1FL*).

Next, we show the encoded formula, called *e1FL*, for the dispatch of external event *e1* – `xChangePrepare` – in step 1 in the following, and the dispatch of the remaining external events could be encoded similarly.

$$\texttt{xChangePrepare\_0} = \textit{false} \wedge$$
$$\texttt{xChangePrepare\_1} = \textit{true} \wedge$$
$$\texttt{x10KYenRequest\_1} = \texttt{x10KYenRequest\_0} \wedge \ldots\ldots \wedge$$
$$\texttt{retStatus\_1} = \texttt{retStatus\_0}$$

where the equation in the first line in *e1FL* characterizes the execution condition, and the equations in the remaining lines characterize the execution effects. Again we omitted some equations for those variables whose values are unchanged.

The entire encoded formula for execution step 1, called *step1FL*, is written as: *c1FL* $\vee \ldots \vee$ *c7Fl* $\vee$ *e1FL* $\vee \ldots \vee$ *e3FL*, which intuitively means that one of the normal cells or dispatch of an external event is possible to be executed in step 1 from the initial state. Formulas for other execution steps within the given bound, say *k*, could be encoded similarly, while note that in each step a fresh set of new variables are introduced and used. Consequentially, all the states of the execution steps within *k* is expressed by an *and*-conjunction of the form *initFL* $\wedge$ *step1FL* $\wedge \ldots \wedge$ *stepkFL*. The part of *Loops on Steps within Upper Bound* written in Figure 2 is processed following the way as we have just explained above.

The encoding rules implemented in Garakabu2 for Linear Temporal Logical (LTL) properties are similar to the approach described in [10]. We have implemented efficient algorithms for generation of Negation Normal Form (NNF) for LTL formulas and their encoding, which will be reported in another opportunity.

3.2 BMC of STM Designs

Based on the encoding approach introduced in the previous subsection, Garakabu2 encodes a STM design and negation of a user-specified LTL property into a logical formula. The formula is then input into a state-of-the-art SMT solver – CVC3 [11] that are incorporated in Garakabu2, to try to detect counterexamples (i.e., design errors) through determination of the formula's satisfiability.

step 0	*Initial state of MCS*
step 1	*Event xChangePrepare is dispatched by the environment*
step 2	*Cell (0,0) of CHANGER is executed*
step 3	*Event x10KYenRequest is dispatched by the environment*
step 4	*Lhs of Cell (1,1) of CHANGER is executed*
step 5	*Cell (0,0) of RETURNER is executed*
step 6	*Event xReceive is dispatched by the environment*
step 7	*Lhs of Cell (1,1) of RETURNER is executed*
step 8	*Cell (2,2) of CHANGER is executed*
step 9	*Event x10KYenRequest is dispatched by the environment*
step 10	*Rhs of Cell (1,1) of CHANGER is executed*
step 11	*Cell (0,0) of RETURNER is executed*
step 12	*Event xReceive is dispatched by the environment*
step 13	*Rhs of Cell (1,1) of RETURNER is executed*

Fig. 3: Execution Sequence of the Counterexample w.r.t. an Invalid Cell

In this paper, we show an example LTL property that is used to check whether the invalid cell `taken-WAIT_REQUEST` of MCS is unreachable, i.e., whether the event `taken` is indeed not possible to be dispatched when CHANGER is in the status `WAIT_REQUEST`. The property is declared by using the LTL operator *globally* [2] – denoted by the symbol `[G]` in Garakabu2 – as follows:

$$[G](\neg(\texttt{taken} = \textit{true} \wedge \texttt{chgStatus} = 1))$$

which could be read intuitively as that the two equations could not hold at the same time for any executions of MCS. We input this LTL property and the STM design MCS developed using ZIPC into Garakabu2. Garakabu2 finds in 2.8 seconds a counterexample whose execution step is 13. Figure 3 shows execution sequence of the counterexample. We use *event* and *status* index numbers (listed in Figure 1) to pinpoint a cell, e.g., "Cell (0,0)" of CHANGER denotes the cell `xChangePrepare-IDEL`, and "Lhs" or "Rhs" denotes the concrete sub-cell of a specified cell.

After observing the execution sequence in Figure 3, we could understand that the problem (error) occurs in the Cell (1, 1), which dispatches the event `paid` even if there is not enough balance. We therefore revised MCS by removing the second assignment of this cell and checked the LTL property again. This time no counterexample is found when the upper bound is set to 50.

However, it is worth pointing out that abstracting the clear execution sequence in Figure 3 from the output (that represents a counterexample) of CVC3 is extremely cumbersome since SMT solvers including CVC3 just simply output a set of variable assignments that makes the input formula *true*. Due to this, understanding a counterexample becomes a tough task, especially for on-site software engineers.

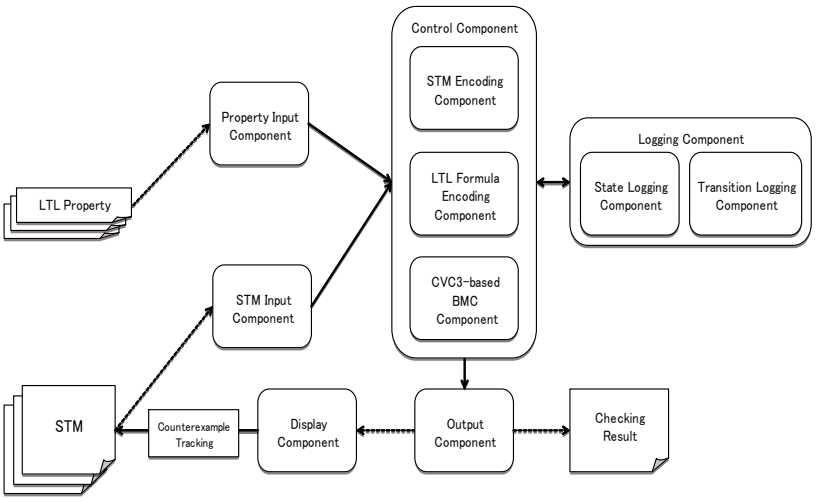


Fig. 4: High-level Structure of Garakabu2

4. Attempts for Enhancing Usability

Although formal verification techniques, especially model checking, have been broadly recognized to be effective for enhancing software reliability, they have not been adopted as a standard phase of the software development process, and the use of them in a practical setting is still rare. A significant barrier behind, as also often mentioned in the literature, is that formal verification techniques are hard to learn and apply since a sufficiently enough mathematical knowledge is required and necessary, which however, is difficult for on-site software engineers.

In this paper, we focus only on model checking among other formal verification techniques. Based on our observation of Japanese software industry, the above mentioned barrier is further divided into two detailed (sub-)barriers, while one is related to the *input* and the other is related to the *output* of model checking tools. In the following, we discuss our attempts made in Garakabu2 for alleviating separately the two (sub-)barriers, which we hope to be helpful or in a direction that may be helpful for promoting practical use of model checking techniques for on-site software development.

4.1 Input – Table-based STM Design

Regarding input, to apply model checking techniques and tools to analyze a target design, software engineers have to firstly describe the (parts of the) design with a model description language that is specific to the model checker to be used[1]. However, fully mastering a model description language is non-trivial, which consequentially makes it difficult to guarantee the consistency between the original design and re-described one, i.e., whether the re-described one has exactly the same semantics as the original one.

To alleviate this barrier, Garakabu2 accepts as its input models the designs developed using STM for on-site software development, i.e., the STM designs could be model checked just as they are, and therefore there is no need for software engineers to learn a "second" model description language. In addition, we believe that the *table-based* feature of the STM language also makes itself easier to master compared to *program-like* model description languages usually used by most of state-of-the-art model checkers.

Figure 4 shows a high-level component structure of Garakabu2. To conduct BMC with Garakabu2, software engineers are only responsible to develop a STM design (that is to be used for later software development), LTL properties to be checked against the design, and an upper bound number to which depth the design is to be checked. The difficult mathematical computation for BMC (the *Control Component* in Figure 4) like those described in Section 3 are hidden into Garakabu2. Note however that, we still provide *Logging Component* in Garakabu2, which registers the states and transitions information that are generated and used inside. These information could be examined by advanced users when necessary if a fully understanding on how Garakabu2 works for their problems. The *output* and *display* components are to be discussed in the next subsection.

4.2 Output – Counterexample Tracking

Regarding output, as been discussed in Section 3, abstracting a clear execution sequence from the output of SMT solvers that represents a counterexample is extremely cumbersome. Therefore, understanding a counterexample and consequentially why a design error occurs, also becomes a tough task for software engineers.

[1] Although model checkers such as CBMC are available that could be used directly to analyze software source code, we focus on model checking of software designs since we believe that quite a lot source code errors are due to design errors, as discussed in Section 1.

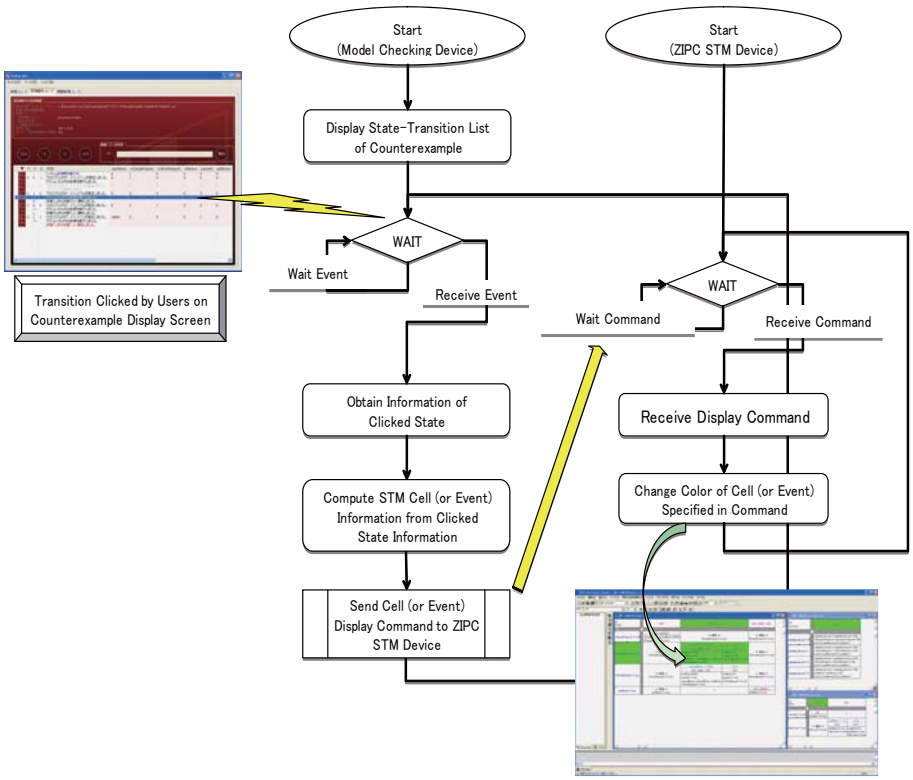


Fig. 5: Graphically Counterexample Tracking with Garakabu2 and ZIPC

To alleviate this barrier, we implemented in Garakabu2 an *output* component that computes the execution sequences of counterexamples from CVC3's results (sets of variable assignments), and a *display* component that displays graphically (in the form of table) the execution sequences in STM designs (tables) in the ZIPC environment.

Figure 5 shows the process of graphically tracking the execution sequences of counterexamples, where the sub-process to the left shows the activities executed by Garakabu2 and the sub-process to the right shows the activities executed by ZIPC environment. When Garakabu2 discovered a counterexample through BMC, the execution sequence of the counterexample is computed and listed in Garakabu2's output interface (as shown in the lower part of the sub-figure in the top-left of Figure 5). Users of Garakabu2 could click each segment of the sequence (possibly from top to bottom), and such click-activities are converted into ZIPC recognizable commands and transferred to ZIPC. ZIPC then receives and parses these commands and highlight the corresponding event or cell of a STM by changing its color, indicating the event or cell executed in that clicked execution step (as shown in the sub-figure in the bottom-right of Figure5). In addition, all the variables defined and used by users in the STM design, together with their respective values in each execution step, are listed in Garakabu2's output interface. Such a graphical tracking functionality is believed by us to be greatly helpful for users of Garakabu2 (software engineers) to understand the exact reasons of counterexamples.

5. Conclusions and Future Work

In this paper, we have introduced the BMC techniques implemented in Garakabu2 for model checking of software designs developed using a table-based language – STM. Additionally, our attempts for enhancing the usabilities of model checking techniques in general and Garakabu2 in particular have also been described. The effectiveness of formal verification techniques for enhancing software reliability has been recognized broadly and the importance of them has been recently stressed again in international standards such as IEC61508 [12] and ISO26262 [13]. We believe that *visibility* is a key issue for usability. Based on this view point, we expect that our work could be helpful to some extent for promoting the practical use of formal verification techniques in on-site software development and finally making it become a standard phase of software development process.

There are still quite a lot to be done as future work. The precise meaning of LTL formulas is commonly known to be non-intuitive and can confound even the experts [3]. Therefore specifying properties to be checked with LTL can be hard for software engineers. We are currently developing in Garakabu2 a graphical editor that could help software engineers specify LTL properties with a set of predefined, often-

used, and domain-specific property patterns [14], and/or with a set of graphical notations for LTL operators [15]. In addition, to make it possible for BMC to find counterexamples of deep execution steps, i.e., expand the state space that could be checked, we are currently investigating approaches that could take advantages of recent advances of multi-core processors and large-scale computing clusters (including distributed data processing technique/framework – Hadoop etc).

Acknowledgment

This research is conducted as a program for the "Regional Innovation Cluster (Global Type, the 2nd Stage)" by Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japan. We would like to thank all relevant organizations and people for their support.

References

[1] CATS Co., Ltd., Japan, *ZIPC Version 9.2*. URL: www.zipc.com.

[2] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*, MIT Press, 1999.

[3] G. Holzmann, *The SPIN Model Checker: Primer and Reference Manual*, ISBN 0-321-22862-6, Addison-Wesley, 2008.

[4] E. Clarke, D. Kroening, and F. Lerda, *A Tool for Checking ANSI-C Programs*, In TACAS 2004, LNCS 2988, pp. 168-176, Springer, 2004.

[5] A. Biere, A. Cimatti, E.M. Clarke and Y. Zhu, *Symbolic Model Checking without BDDs*, In TACAS 1999, LNCS 1579, pp. 193-207, Springer, 1999.

[6] W. Kong, T. Shiraishi, N. Katahira, M. Watanabe, T. Katayama, A. Fukuda, *An SMT-based Approach to Bounded Model Checking of Designs in State Transition Matrix*, To appear in IEICE Transactions on Information and Systems, 94-D(5), 2011.

[7] T. Shiraishi, W. Kong, Y. Mizushima, N. Katahira, M. Matsumoto, M. Watanabe, T. Katayama, A. Fukuda, *Model Checking of Software Design in State Transition Matrix*, In SERP 2010, pp. 507-513, 2010.

[8] J. Dubrovin. *Checking bounded reachability in asynchronous systems by symbolic event tracing*. Technical Report TKK-ICS-R14, Helsinki University of Technology, 2009.

[9] C. Barrett, R. Sebastiani, S. Seshia, and C. Tinelli, *Satisfiability Modulo Theories*, In Book: *Handbook of Satisfiability*. Edited by A. Biere, M. Heule, H. Maaren, and T. Walsh, ISBN 978-1-58603-929-5, IOS Press, 2009.

[10] T. Latvala, A. Biere, K. Heljanko, and T. Junttila, *Simple Bounded LTL Model Checking*, In FMCAD 2004, LNCS 3312, pp. 186-200, Springer, 2005.

[11] C. Barrett and C. Tinelli, *CVC3*, In CAV 2007, LNCS 4590, pp. 298-302, Springer, 2007.

[12] 61508 Association, *International Standard for Electrical, Electronic and Programmable Electronic Safety related Systems*, URL: http://www.61508.org.

[13] IOS, *Road vehicles – Functional Safety* (under development), URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=43464.

[14] M. Dwyer, G. Avrunin, J. Corbett, *Patterns in Property Specifications for Finite-State Verification*, In ICSE 1998, pp. 411-420, ACM, 1998.

[15] S. Koike, S. Yoshida, and H. Ohsaki, *Diagrammatic Notation for LTL Model Checking*, In FOSE 2007, pp. 35-44, 2007.

Agile Methodology for Designing and Improving Enterprise Scenarios[1]

Ł. Budnik and H. Krawczyk
Computer Architecture Department, Gdańsk University of Technology, Gdańsk, Poland

Abstract - *An agile methodology for designing and improving enterprise business scenarios is proposed. It provides development milestones, fine-grained metrics, and improvement procedures which allow improving scenario performance, quality, and usability. A simple case study is considered and evaluation is discussed. An example implementation of DIES methodology – the DIES system is presented. Finally, authors present the DIES system as an assessment and evaluation tool for cluster management environment and C-coded processes.*

Keywords: business processes, enterprise scenarios, quality monitoring, quality analysis, quality improvement, development methodology

1 Introduction

Services are the basic building blocks of distributed business scenarios. With the advance of SOA services gained a new momentum. In an SOA environment, new services can be dynamically updated, replaced, or removed. This brings many possibilities, but also carries a high risk of unsatisfactory quality and performance metrics of business scenarios in a dynamic services environment.

1.1 Designing business scenarios – problems and needs

Designing a business scenario is not an easy task. Implementation of human endeavor in a broader perspective is a coordinated cooperation of many organizations in order to achieve the collective success of the enterprise. It must be a joint cooperation of process engineers, business analysts, end users, and the management team.

The main problem encountered by organizations, and in consequence of the fact that the transformation between problem domain and IT domain is a difficult task, is the problem of defining, monitoring, enforcing, and evaluating quality and performance of the implemented business process scenario. Each service, and the entire business scenario, must meet strict quality criteria agreed by the parties in the contract. Combining services can increase productivity and reliability of the solution, but also carries a risk of failure to comply with business objectives. The owner of a business process must make the optimal choice of services and monitor them on an ongoing basis to maintain the highest possible quality for its customers (Quality of Service, QoS). Software engineers and business analysts while creating a new business scenario must make a choice of alternative services and various vendors. To make a proper choice static analysis, in a dynamic Web services environment, does not always guarantee success. Over time, the environment may change and the process may not satisfy the initial conditions. For example, quality and performance parameters may change due to the increased popularity of one of the services, which may result in a longer response time, or an increased license fee. Thus, the provider of business process must not only make the optimal choice of services, but also constantly monitor and respond to all undesirable situations so that the quality remains at a sufficiently high level. Also, having used external components, the owner of the business scenario must accept the fact that s/he might not have any control over such components.

Also, process and software engineers, must constantly monitor the business scenario in order to detect any undesirable situations like a drop in quality and performance metrics. This issue may be addressed by providing a dynamic feedback and alerts mechanism which warns engineers if the business scenario as a whole, or any of its complementary services, do not meet strict quality criteria. Such mechanism should also offer detection of unexpected drops in the quality criteria. For example it has been accepted that a service over a few months time may increase its original execution time by 50%. If the execution time has increased by 10% during all previous runs, but in the current run increased by 35%, the 50% threshold is not yet exceeded, but within a single scenario run, the value increased unexpectedly rapidly. Such situations should be detected and reported accordingly.

Finally, functional requirements such as data encryption or authentication method can be expressed in the WS-Policy notation [16]. Note, however, that the non-functional requirements, and in particular [17]:

[1] This work was realized as a part of MAYDAY EURO 2012 project, Operational Program Innovative Economy 2007-2013, Priority 2, Infrastructure area R&D

- performance requirements,
- regulations,
- business guidelines
- enterprise policies,
- various restrictions,

are either difficult, or impossible to express in WS-Policy. Hence, few scenarios define any policies and quality and performance requirements [18]. QoS-aware composition of services is also seen as a research challenge for the future [19].

As our experiments show, the vast majority of processing time and almost all issues are related to remote invocations of services that is why the most important task in implementing business scenario is the evaluation of business scenario's structure and making the optimal choice of services. In the paper we address business scenario development issues. We propose a new continuous development methodology which introduces flexibility and agility into development of business scenarios. The proposed methodology is focused on quality and performance evaluation, storing and analyzing partial results, and providing feedback to the end user. In the paper we also present a DIES system which fully supports the proposed methodology and was successfully used as an evaluation and assessment tool of enterprise business scenarios.

2 DIES Development Methodology

After the credit crunch companies all around the world started to cut costs. Now, in order to gain advantage over competitors and be ahead of them when customers will start to buy goods and services again, they have to prepare themselves and be able to react to market changes faster. The waterfall approach is in decline and agile methodologies are becoming more and more popular. We undertook an effort to bring agility and flexibility into designing executable enterprise business scenarios. Our aim was to create a universal, standard- and vendor-agnostic agile methodology for developing business scenarios.

First, we defined a business scenario development as a continuous ongoing process which is depicted in Fig. 1. The key concept that distinguishes our approach from other similar approaches, apart from it being a continuous approach, is the execution phase where we run instrumented versions of business scenarios which allows us to gather partial results and quality and performance metrics. Metrics which we collect during the execution of a scenario are grouped into services- and scenario-specific groups. In addition, our methodology offers a direct mapping of metrics groups into improvement procedures which helps to make decisions of what actions should be taken when quality and performance metrics are unsatisfactory. These concepts are explained in detail in the following Sections.

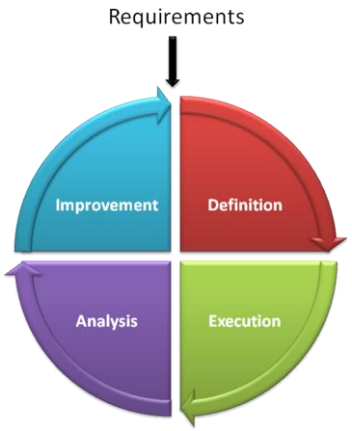


Figure 1. The idea of agile business scenario development methodology

2.1 DIES Development Methodology

The experiments we conducted over the last few years and the experience we gained allowed us to define Design and Improvement of Enterprise Scenarios (DIES) development methodology. The proposed development methodology assumes that the business scenario technology must fulfill the following requirements:

- Definition of scenario needs to be extensible (e.g., defined in XML),
- Technology must support invocation of remote monitoring services.

Below we present milestones in business scenario development life cycle which guarantee high quality of the end product. The proposed DIES development methodology is presented in Fig. 2.

2.1.1 Formal Description of Business Scenario

A written form describing a business scenario. A formal description may not consist of only a requirement specification document, it may consist of a contract which specifies project dates and budget; legislative regulatory requirements are also often a part of formal descriptions of a business scenario.

2.1.2 Visual Representation of Business Scenario

Defined by the business owner, this ought to be a conceptual, easy to understand diagram highlighting the most important aspects of the business scenario. Because our methodology is standard-agnostic we do not impose any notation. Depending on the business owner preferences it may be a graph, an UML diagram, or SOA-dedicated BPMN diagram [14].

2.1.3 Abstract Model of Business Scenario

An abstract process expressed in the final business scenario's notation. The abstract process defines services

composition, constraints, and the general flow of control. The simplified abstract model should be understandable by the business owner. WS-BPEL [3][6] is an excellent example of a technology that facilitates both features of abstract model and executable scenario. WS-BPEL abstract model can be easily turned into executable scenario which is still understandable by the business owner.

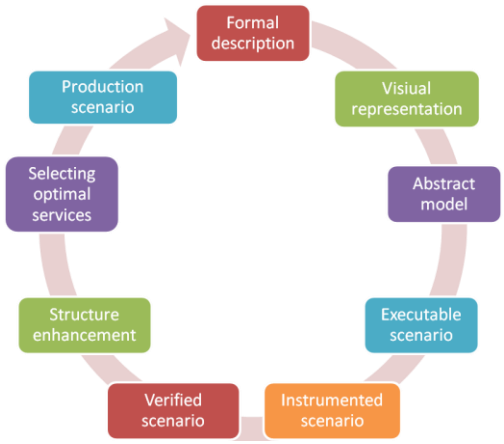


Figure 2. DIES development methodology

2.1.4 Executable scenario

Executable scenario is a detailed implementation of an abstract scenario. A detailed implementation consists of:

1. Local instructions – used to manipulate data, handle errors, express compensation actions, and control the flow of the business scenario; often referred to as programming in the small;
2. Invocations of external services – used to invoke remote services, e.g. sending/reading data to/from queues or invoking Web Services; often referred to as programming in the large.

Apart from defining complete business logic, executable scenario contains deployment descriptors and other platform- and vendor-specific artifacts.

2.1.5 Instrumented scenario

Instrumentation process adds a monitor service (and depending on the technology used, its artifacts), the logic of events logging, and most importantly, constructs which allow the scenario to set and verify the execution context. Instrumentation is fully transparent to the end user and its result is still a correct executable business scenario. Thanks to the instrumentation process the business scenario is not a black box any more as all of the internal services invocations are monitored.

2.1.6 Verified scenario

Verified scenario is a stage in DIES development methodology in which the usability of the scenario has been proven. Verification usually comprises of running a several test cases and comparing actual returned data with that which were expected.

2.1.7 Structure enhancement

Once the scenario has been verified to work correctly against the requirements, DIES evaluation and analysis takes place. The first optimization procedure is enhancing scenario's structure. This comprises of removing loops, shortening their lengths, introducing EIP (e.g., aggregator pattern), reordering services invocations, error handling, and compensation scopes. In this step business scenario-specific DIES metrics are used. Usually this improvement technique is cheap in terms of time and budget as it only requires evaluation of scenario structure.

2.1.8 Selecting optimal services

After structure enhancement has been applied selecting optimal services improvement procedure is applied. Having selected the optimal structure a more detailed evaluation of services is performed. From the set of alternative services only the best ones are selected. In this step a service-specific DIES metrics are used. This improvement procedure is more expensive than structure enhancement and should be carried out if the first technique does not yield the expected results. Implementing this technique requires additional development time as alternative sets of scenarios must be designed, implemented, and evaluated. Also, this technique requires additional IT resources like introducing a services repository which would store services definitions and their quality and performance history records.

2.1.9 Production scenario

Having completed all the above mentioned steps, a business scenario is ready to be deployed to production environment. The quality and performance evaluation should be carried out on an ongoing basis as an integral part of the maintenance phase. Also, we strongly suggest that apart from evaluating quality and performance metrics the system must implement an alerts mechanism to warn software engineers and business analysts when computed scores would be unsatisfactory.

Production scenario through the quality and performance analysis provides a feedback to the business owner which may be an input for new structure optimization, or if services quality drops, may be an input for new selection of services. Also, there might be external factors like change in requirements or change in legal rules which may cause the whole development methodology to start again from the formal description of business scenario stage. All these cases are illustrated in Fig. 2.

2.2 DIES Quality and Performance Metrics

During our research and experiments we prepared a list of quality and performance metrics that we found relevant to evaluating business scenarios. In addition, we introduced concepts of mandatory global metrics as well as optional provider- and operation-level metrics which are used to fine-grain scenario parameters. The following service-specific metrics were chosen [15]:

1. Cost,
2. Security,
3. Effort,
4. Reputation,
5. Execution time,
6. Geographic distance from scenario's node,
7. Size of the input data,
8. Size of the output data,
9. Additional supported WS-* standards.

Apart from service-specific metrics we defined the following business scenario-specific metrics:

1. Number of services invocations,
2. Services invocation processing time,
3. Percentage of hits/hits coverage,
4. Number of loops and lengths of loops.

These are metrics which are gathered by DIES system automatically and which assist in business scenario quality evaluation and improvement.

2.3 Improvement Procedures

We identified four main areas which may lead to unsatisfactory quality and performance score. These factors are:

1. Weak points in scenario design,
2. Poor quality of services/insufficient number of alternative services,
3. Bottlenecks in network infrastructure,
4. Insufficient knowledge about the scenario itself.

To assist software engineers in improvement of business scenarios, we defined a set of procedures which target each area. In experiments which we conducted, the following procedures turned out to be very valuable and all are supported by the DIES system. We grouped them into 4 groups.

2.3.1 Structure enhancement procedures

The application of these procedures is indicated by unsatisfying business scenario-specific metrics:

- Minimizing the number of invocation loops and/or their lengths,
- Introducing EIP patterns [9] and minimizing the number of services invocations – aggregator pattern turned out to be a very successful technique,
- Replacing programming in the large by programming in the small, i.e. replacing remote invocation of services by a logic embedded directly into business scenario,
- Reordering invocations of services (which also affects invocations of services in compensation and fault handlers).

2.3.2 Selecting optimal services

The following improvement procedures are all related to service-specific metrics:

- Evaluating alternative services in order to select the best services,
- Replacing data value objects with identifiers wherever possible to reduce the size of exchanged data.

2.3.3 Scaling out

In the era of cloud computing it might be tempting to try to resolve quality and performance issues by scaling out. However, this technique should be implemented as a last resort as at some stage adding a new box will not present improvements and on the contrary may further decrease them.

2.3.4 Analyzing partial results

The last procedure group consists of gathering and analyzing partial results technique. This technique does not yield results in the short term but is beneficial in the longer term. This technique helps process and software engineers in designing and implementing new scenarios based on current business scenarios. Maintaining, managing, and recording partial results of scenarios not only increases engineers' experience and skill sets, but also creates a foundation for future test cases and simulations. Gathered data may be an input to data mining research which may improve the quality of service compositions [13]. Also, the simulation process is a very important technique which helps in designing effective business solutions. Using simulations, engineers may test the impact of new requirements without expensive and time consuming development effort, conduct stress tests, or even estimate most probable final quality and performance metrics.

3 Support for the DIES Methodology

The system called MS DIES (Measurement System DIES) has been designed and developed. Its main objective is to help software engineers and business analysts to maintain the quality and performance of enterprise business scenarios. The DIES system provides an abstraction layer which completely hides the underlying business scenario technology from the business user. Further, DIES provides a mechanism which makes it possible to support any business scenario technology.

DIES system implements fully our agile methodology. It supports basic business scenario lifecycle management including instrumentation, defining quality and performance metrics and weights on all three granularity levels (global-, provider-, and operation-level metrics and weights). Also, DIES gathers partial results and computes quality and performance scores. Additionally, DIES supports the generation of alerts when quality and performance scores are below defined thresholds. Finally, DIES' Web Console provides a wide array of statistical views e.g., tabular data, charts, or graphs.

For the purpose of our experiments we implemented support for WS-BPEL standard and Apache ODE [7] as its execution engine.

3.1 General Architecture

The logical architecture of DIES in the context of running scenarios and gathering partial results and computing quality and performance scores is shown in Fig. 3. The figure, apart from the control flow, shows the six main subsystems of the DIES system.

3.1.1 Web Console

Provides a convenient web interface for managing business scenarios. The business user need not know any of the specific operations related to the lifecycle management, or know the API for remote execution of scenarios. The status of the scenario, its instances, data collected during the execution of scenario instance, and evaluation of the results of quality and performance metrics are all available from the Web Console.

3.1.2 Engine

The central part of the whole system. Engine provides a mechanism for meta data by which the business user can describe a scenario, define its parameters (quality factors and weights), or configure monitoring levels. Based on the meta data attached to the scenario engine extends the definition of the scenario. The process of enhancement and transformation of scenarios is transparent to the end user. In addition, the DIES system implements a flexible and unified API for lifecycle management, monitoring quality and performance parameters, and the collection of partial results. Thanks to this DIES can support many different standards and technologies of business scenarios. Also, the engine offers an API to run scenarios remotely, allowing a broader integration with external systems in an SOA manner.

3.1.3 Local and DIES Monitor

Local monitor as an argument takes basic quality metrics and logged messages and asynchronously passes them to DIES monitor. This module has been introduced in order to facilitate business scenario technologies that do not support asynchronous invocations, or technologies where the processing overhead of correlating such invocations is very expensive. This module is optional. DIES Monitor, then, retrieves the data from Local Monitor (if used), and logs the parameters of Web Services and saves the partial results in database. The collected data are transferred to the Analyzer module.

3.1.4 Analyzer

Analyzes the data gathered by the DIES Monitor. If the parameters do not comply with values defined by process engineers, it generates reports and alerts.

3.1.5 Database

All the parameters, results of the computations of the quality function, partial results, alerts, and reports are stored in IBM DB2 pureXML [10] database. XML database is especially suited for storing partial results which are XML documents.

4 Case Study

Because of our project's legal restrictions the names and functions of services participating in business scenario has been altered. However, all invocations and flow of control remain valid. For the purpose of the case study we adopted a warehouse and products orders use case.

4.1 Scenario and its definition

A large warehouse had an online B2B system for ordering goods for its largest clients (local warehouses). As a response to the dynamically changing market, the board decided to create a B2C channel as well. The current system handles 500 requests a day, but market analysis shows that a B2C channel would handle even up to 10 000 requests a day which is 2 000% growth. Before launching the B2C and investing money in an IT infrastructure upgrade, it was decided to improve the business scenario first.

The semi-formal scenario's description looked like this:

1. Business scenario is run with the following arguments: client data and a list of products that he wishes to order;
2. The business scenario iterates through the product list and queries all three warehouses:
    a. Building materials warehouse,
    b. Interior materials warehouse,
    c. Electrical and Security warehouse;
3. List of results is processed:
    a. if at least one of the products is not found, process ends with a relevant message,
    b. if a quantity of at least one of the products is not enough then:

i. Internal supply service is invoked with the ran-out product,
ii. Process ends with a relevant message;

c. If all products are found and their quantity is sufficient, business scenario continues;

4. Proper warehouses booking methods are invoked, warehouse service returns booking ID;
5. Warehouse data, booking ID and client data is then sent to the logistic service;
6. Account service is then invoked with order reference;
   a. If an unexpected error occurs, exception is caught, and as a consequence a compensation handler is invoked,
   b. Compensation methods on warehouses services are invoked
   c. Compensation method on logistic service is invoked
7. Finally, if account service completes without any errors, loyalty program service is invoked and the customer's loyalty program points are assigned.

As part of the analysis, the scenario was ran on the sample of 12 orders grouped in 4 test cases. The first test set (25% of all test scenarios) represented the original B2B channel – large customers ordering 20 and more products. The rest (75% of test scenarios) were targeted at the new B2C channel because, after the business strategy change, this channel will be the main branch of the online business.

4.2 Improvement

The experiment for large orders showed a significant number of warehouse service invocations. For example, an order of 22 products in the most pessimistic case resulted in 66 warehouse services invocations – all 3 warehouse services were queried for 22 products. As an improvement it was suggested to use an aggregator enterprise integration pattern. Instead of looping through products list and checking each product's availability the whole list was sent to every warehouse service. The change of the warehouse services turned out to be simple thanks to reusing the existing functionality.

Also, this new approach simplified the business scenario logic, where quite complex WS-BPEL array manipulation constructs were used to detect not found products and to create input data for warehouse booking services.

During review of stored partial results it was spotted that 80% of product orders contained products from building materials warehouse, 15% of orders contained products from interiors warehouse, and only 5% of orders had at least one product item which came from an electronic materials warehouse. Reordering of services invocation improvement procedure was applied. It was decided to reorder the invocation of warehouse services so that the most popular warehouse service is invoked first, and the least popular one is invoked last.

Also, during review of stored partial results it has been spotted that the order of invocation of logistic service and account service should be changed. In current implementation of business scenario logistics service is invoked first and then account service is called. When an exception is thrown by account service shipping actions must be reverted. With the order change, that is account service to be invoked before logistics service, when exception is thrown by account service only warehouse booking actions must be cancelled.

5 Final Remarks

5.1 New Development Methodology

We showed that DIES agile development methodology can be successfully applied in practice. We designed and implemented DIES system which follows all guidelines which we included in proposed methodology.

DIES methodology provides development milestones, fine-grained metrics, and improvement procedures which allow improving scenario performance, quality, and usability and this way allows for greater control over the scenario.

Finally, a unique feature of proposed methodology is the collection of partial results of all services participating in the scenario. Implemented approach allows the collection of both input and output data without modify the external services. With this approach it is possible to store a complete history of the execution of all instances of scenarios. Noteworthy is the fact that the collected partial results may be used as a material for further optimization of the existing scenarios as well as allowing simulation and testing of future scenarios and their services.

5.2 Advantages and disadvantages of the methodology

The key advantage of the proposed methodology is the business scenario-centric approach to data collection and metrics measurement. Thanks to the chosen approach all external Web Services participating in the scenario are seen from a business scenario point of view. An example might be that the logging of Web Service's execution time, value measured by the service provider may be very confusing; the measured time given by the supplier may be 100 ms, but from the viewpoint of the scenario, including transmission time and SOAP processing time can significantly affect and distort the execution time by up to 1 second.

The main disadvantage is that the instrumentation process introduces an additional monitoring service.

However, the negative probe effect was minimized by installing the service on a physical node where the execution engine is installed. The local monitoring service asynchronously sends retrieved data to the DIES monitoring service. The control flow is returned to the business scenario instantly. Also, the instrumented scenario does not perform any computations it only sends gathered results to monitoring service.

5.3 Large possibilities for application

We have successfully applied our methodology to services compositions deployed in other environments.

One of the applications was KASKADA system [11] [12] where we had not only used DIES methodology to develop business scenarios, but also undertook an effort to use DIES system as an assessment and evaluation tool for KASKADA's processes. KASKADA's processes are coded in C language. Thanks to using DIES we gained higher flexibility and agility. During time required to code a C-based business scenario we were able to prepare a set of alternative WS-BPEL scenarios. DIES system, thanks to its capabilities, allowed us to gather partial results as well as quality and performance factors out-of-the-box which helped us improve final scenario and significantly reduce risk of not meeting strict quality and performance criteria.

6 References

[1] B. Steffen, P. Narayan, "Full life-cycle support for end-to-end processes", IEEE Computer, November 2007.

[2] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Web Services – Concepts, Architectures and Applications", Springer Verlag, 2004, ISBN 3-540-44008-9.

[3] B. Margolis, J. Sharpe, "SOA for the Business Developer Concepts, BPEL, and SCA", MC Press, 2007, ISBN 1-58347-065-4.

[4] J. Zdravkovic, M. Henkel, P. Johannesson, "Moving from Business to technology with Service-Based Processes", IEEE Internet Computing, May 2005.

[5] S. Ho Ha, S. Chan Park, "Service Quality Improvement through Business Process Management based on Data Mining", ACM SIGKDD Explorations, Vol. 8, No. 1, June 2006.

[6] A. Alves, A. Arkin, S. Askary, C. Barreto et. al. "Web Services Business Process Execution Language Version 2.0", OASIS Open, 11 April 2007.

[7] The Apache Software Foundation – Apache Orchestration Director Engine, http://ode.apache.org/, 2009.

[8] M. Stal "Web Services: Beyond Component-based Computing", Comunications of the ACM, Vol. 45, No. 10, October 2002.

[9] G. Hohpe, B. Woolf "Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions", Addison-Wesley Professional, 2003, ISBN 0-3212-0068-3.

[10] IBM – IBM DB2 Data management server, http://www-01.ibm.com/software/data/db2/9/features.html.

[11] H. Krawczyk, K. Bańczyk, J. Proficz, "Parallel Processing of Multimedia Streams", in proceedings of XV Konferencja Zastosowania komputerów w elektrotechnice, April 2011.

[12] H. Krawczyk, R. Knopa, J. Proficz, "Basic management strategies on KASKADA platform", in proceedings of EUROCON 2011, April 2011.

[13] S. Bayati, A. F. Nejad, S. Kharazmi, A. Bahreininejad, "Using Association Rule Mining to Improve Sementic Web Services Composition Performance", 2nd International Conference on Computer, Control and Communication (IC4 2009), February 2009.

[14] Object Management Group/Business Process Management Initiative "Business Process Model and Notation 1.2", Object Management Group, http://www.omg.org/spec/BPMN/1.2/.

[15] Ł. Budnik, H. Krawczyk, "Dynamic Analysis of Enterprise Business Scenarios", in proceedings of 5th International Workshop on Advances in Quality of Service Management, August 2011

[16] A. S. Vedamuthu, D. Orchard, F. Hirsch, M. Hondo, "Web Services Policy 1.5 – Framework", World Wide Web Consortium.

[17] M. Kaiser, "Toward the Realization of Policy-Oriented Enterprise Management", IEEE Computer, November 2007

[18] F. Curbera, "Component Contracts in Service-Oriented Architectures", IEEE Computer, November 2007.

[19] M. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", IEEE Computer, November 2007.

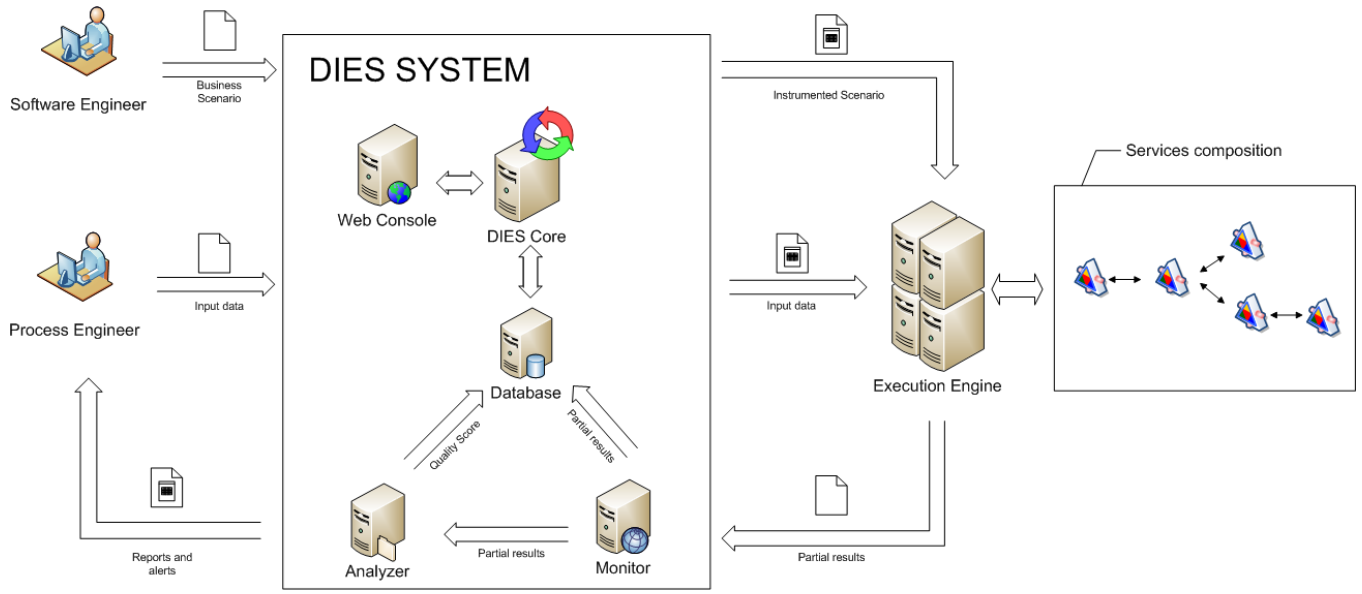


Figure 3. Scenario execution and monitoring using DIES system

A Distributed Chess Playing Software System Model Using Dynamic CPU Availability Prediction

Khondker Shajadul Hasan[1]

[1]School of Computer Science, University of Oklahoma, 110 W. Boyd St., Norman, OK 73019, USA
shajadul@ou.edu

Abstract - *Traditionally chess gaming software takes considerable amount of time in calculating the optimal move that can be made by a computer system. Most computer systems that make such calculations use some version of the minimax search with alpha beta pruning. Minimax with alpha beta pruning can be made distributed with principal variation splitting. The system can thus be used to reduce the time necessary to calculate the optimal move. Additionally, to select the distributed node containing less workload from the grid, a dynamic CPU availability prediction can be deployed to enhance the performance of task (searching game tree to find the optimal move) execution. An algorithm for dynamic monitoring of CPU has been proposed and discussed in this paper besides PV-Splitting minimum and maximum algorithms. Prediction of CPU availability is important in the context of making task assignment and scheduling decisions. Extensive experimental studies and statistical analysis are performed to evaluate the performance improvement of the model using the chess software system which uses the PV-Splitting for task parallelism.*

Keywords - Adversarial Search, Alpha Beta Pruning, Dynamic CPU Availability, Minimax Algorithm, Multi-core processor, Principal Variation Splitting.

1 Introduction

Chess has become very common game and nearly available on any kind of computer. As chess is in existence for very long time, much of that time, it has been played between human players. Only in recent times (relative to the time chess has been played) have computers started to play chess. The idea of a non human playing chess came to realization with the advent of digital computers. Even with the advent of digital computers, the computer could not play chess well. This was mainly due older computers not having the necessary computational power to play chess [1].

With the improvement in computational power and advanced algorithms, computers now have better chess playing software. However, when playing chess on personal computers, it takes a lot of time to calculate its move for advanced levels and with lack of computational power; it is not enough to challenge the best human players. So, computer scientists turned to distributed processing for improving this scenario [2]. Parallel processing allowed a massive increase in computational power which is needed to challenge good chess players. If the time needed for the computer to calculate a move could be cut, then the game against a computer could be more enjoyable. Our objective in this paper is to reduce the time need by a computer to calculate moves. In 1997, IBM's Deep Blue supercomputer created a sensation by defeating then world champion Gary Kasparov. This caused the world to take computer chess seriously. Nowadays, computer chess has become very common. Chess software is available on nearly any kind of computer.

To reduce the immense searching time for calculating moves, distributed computing has been employed. The processing power of two or more computers connected over a LAN was used to reduce time needed to calculate a move. The part of the program that performs the actual computations will run in the background of the various computers being used. This will allow the computers to be used for other, non cpu intensive purposes, while the computation is being done in the background.

It is necessary to divide the task of finding an optimal move for the computer of a grid to make chess distributed. To do this, Principal Variation Splitting, or PV-Splitting has been employed. PV-Splitting is an algorithm that was originally designed for shared memory multiprocessor systems. PV-Splitting had to be changed slightly in this paper to make it run in a distributed system.

2 The Distributed Model

In this paper, distributed computing have been used in which three or more computers connected over a LAN to reduce time needed to calculate a move with high processing power. The model of the distributed

chess system consists of two types of nodes: a single parent node and two or more worker nodes. The parent node divides the job of calculating the optimal move for the computer to make into smaller jobs. These jobs are then assigned to the worker nodes to compute. These jobs are assigned when messages are passed from the main node to the worker nodes using a local area network. The worker nodes are implemented as a Microsoft Windows service. This allows the worker node to run in the background.

The jobs are divided using principal variation splitting. Each job consists of searching the sub tree of a particular node as its root [4]. Different jobs do not have any intersection. This means the sub trees that are searched do not form a directed acyclic graph. Two different jobs with different root nodes do not have anything in common. Thus there is no communication necessary between workers searching different sub trees.

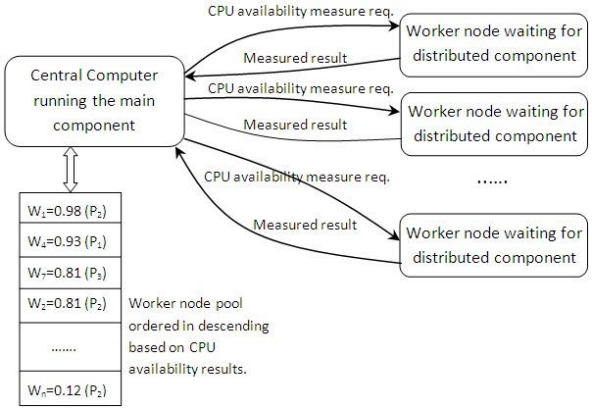


Figure 1: Distributed System Model.

Figure 1 contains the distributed model for the chess game system. The central computer which is running the main component of the game will dynamically measure the CPU availability of each worker node [6]. The measure data is used for arranging the worker nodes into the computer pool. The worker node which has the highest CPU availability gets the highest priority of achieving task as it can process early.

Each sub tree is searched using the minimax [1] algorithm with alpha-beta [2] pruning. When a job is assigned, the alpha beta value of the sub tree's root node's parent node is used. Once a job is finished, the min or max value of the sub tree's root node is returned as a message. The alpha beta values of the parent are then updated. Since there is only one main node in our distributed system, it will reside in one computer. The many worker nodes may reside in more than one computer. There may be a main node and one or more worker nodes on the same computer. There may also be more than one worker node per computer. This can be used for performance gains for computers with multiple CPUs or multiple cores.

Also, all the worker nodes can reside on a single computer. The system will not be distributed, but will be able to take advantages of multi-CPU or multi-core processors. It will also not suffer from any network lag.

3 Principal Variation Splitting

Principal Variation Splitting [3, 4] or PV Splitting was originally designed for multi-processor systems. In this case, it was adapted to work on a multi computer distributed system. PV Splitting is ideal for our model of the distributed system, as PV Splitting allows the job to be divided into smaller jobs from one main node and then have the job assigned to different workers.

The PV Splitting searches the same tree as the minimax algorithm [4]. It does the search in a parallel fashion, or in this case using multiple computers in a distributed system. PV Splitting is designed for the case when alpha beta pruning is used along with minimax search. In PV Splitting, starting with the child nodes of the root node, the leftmost child node's sub tree is searched first and then the remaining child nodes are then searched. This is done recursively. That is when searching the leftmost sub tree, the leftmost child node is expanded into its children first. Among its children, the leftmost child's sub tree is again searched first before searching the remaining children in parallel.

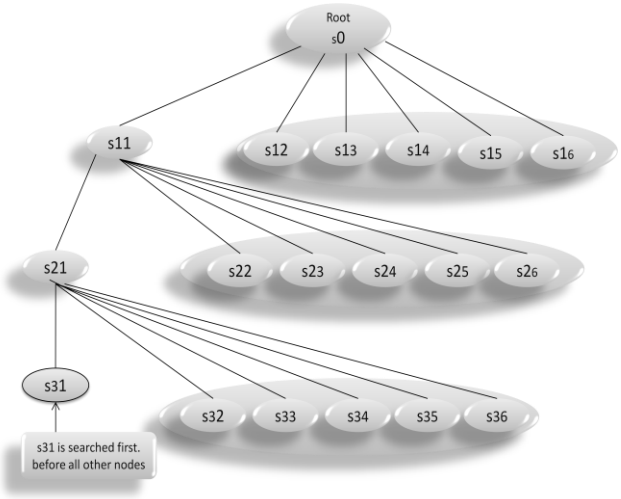


Figure 2: Principal Variation Splitting Procedure.

This is illustrated in Figure 2. The root node s0 is expanded into its children s11, s12, s13, s14, s15, s16. The left sub tree, s11 and all its descendants are searched first before the other children of s0 are

searched in parallel. The node s11 is expanded into s21, s22, s23, s24, s25, s26. Again, s21 and its descendants are searched before the remaining children are searched in parallel. The s21 node is then expanded into its children s31, s32, s33, s34, s35, s36. The node s31 is searched first before all other nodes.

Once the search of the sub tree of s31 has been finished, then sub trees of its sibling nodes s32, s33, s34, s35, s36 are searched in parallel. Similarly, once the search of s21 is complete the siblings of s21: s22, s23, s24, s25 and s26 are then searched in parallel and so on. Finally, when these searches are complete, their parent s0 has been searched completely. Thus the entire tree has been searched. All searches of a sub tree are done with minimax and alpha beta pruning.

Here is the proposed PV-Splitting [5] algorithm adapted for distributed systems:

```
Function PVSplit (start_state,)
Begin
    r := PVSplit_Max(start_state, -infinity, +infinity)
    return r
End
```

```
Function PVSplit_MAX(state, alpha, beta)
Begin
    IF (state is a leaf node)
        return evaluation_function(state)
        state[0]  = first child of state
        IF(current depth < max PV-Split Depth)
            r = PVSplit_MAX(state, alpha, beta)
        Else
             r = MAX(state, alpha, beta)
            alpha = r
            max = r
            state[1..N] = children of state, not including
                          state[0]
            For i = 1 to n in parallel
                (Wait for a worker)
                r = MIN(s[i], alpha, beta) /*MIN
                being minimax algorithms min executed on
                a worker */
                        IF r > max
                                max = r
                        IF r >= beta
                                prune tree
                        IF r > alpha
                                alpha = r
    return max
End
```

Figure 3: Algorithm of PV-Split Max modified for distributed system.

Next function, PVSplit_MIN algorithm for distributed system is given below:

```
Function PVSplit_MIN(state, alpha, beta)
Begin
    IF (state is a leaf node)
        return evaluation_function(state)
        state[0]  = first child of state
        IF(current depth < max PV-Split Depth)
         r = PVSplit_MAX(state, alpha, beta)
         ELSE
             r = MAX(state, alpha, beta)
             beta = r
             min = r
             state[1..N]  = children of state, not
               including state[0]
             For i = 1 to n in parallel
                (Wait for a worker)
        r = MAX(s[i], alpha, beta)  /*MAX
        being  minimax algorithms min executed on a
        lightly loaded worker */
                    IF r < min
                            min = r
                    IF r <= alpha
                            prune tree
                    IF  r < beta
                            beta = r
    return min
End
```

Figure 4: Algorithm of PV-Split Min modified for distributed system.

Each sibling in a parallel search [3] is searched using a worker. Each worker has the ability to search a sub tree using minimax and alpha beta pruning. When the parallel search is started, each sibling node attempts to acquire a free worker. To make sure that sibling nodes do not attempt acquire more workers than exists in the system, a semaphore is used. Once a node acquires a worker, it instructs the worker (through message passing) to search the sub tree of its node. Once the search is done, the worker returns the min/max value for the node as well as other search statistics using message passing. Using the min/max value of the sibling node, the min/max value of the parents is updated as well the alpha beta values.

One of the shortcomings of PV Splitting occurs when the number of worker nodes is greater than the number of sibling nodes that are to be searched. Some of the workers will remain idle if this were the case. The average branching factor for chess is over 30. So on average, for over 30 workers, some of the worker would remain idle.

4 The Chess System Model

A distributed chess game which can select worker nodes for processing its task (searching game tree for optimal move) is modeled using the CPU availability prediction. In this system, distributed computing have been used in which three or more computers connected over a LAN to reduce time needed to calculate a move with high processing power. The part of the program that performs the actual computations will run in the background of the various computers being used. This will allow the computers to be used for other, non CPU intensive processes.

A static model can be deployed to predict the CPU availability for distributed environment. Static prediction model depends on prior information about the number of CPU bound tasks *(n)* in the run queue, the number of I/O bound task *(m)* in the run-queue, and the unloaded CPU usage ($\hat{U}_i$) value for these last processes [7]. As I/O bound processes cannot use the processor all of this time, the CPU-bound processes steal them for some fraction of their quantums. On average, this behavior is coherent with the Linux scheduler mechanisms. The Linux scheduler provides higher priority to I/O bound process than CPU bound process by assigning I/O bound process in higher priority queue.

If the number of CPU core is *c*, the number of hardware thread is *HT*, and the number of run-queue is *R* then simply *R = (c*HT).* That is, for each logical thread (hyper-thread) there is a separate private run-queue. As the static model derived for measuring CPU availability is based on the prior information about the run-queue, for a dual-core processor with two hyper-threading requires four separate predictions (one for each logical processor). Similarly, for a multi-core system, for each logical processor we need to measure separate CPU availability predictions. Therefore, the definitive expressions for the static prediction model to compute the CPU assignment prediction for a new process, denoted by α_{lp} where $lp \in \{1, 2, ..., R\}$, is:

$$\alpha_{lp} = \begin{cases} \frac{1-\sum_{i=1}^{m} U_{i,lp}}{n+1} & if \ \frac{1-\sum_{i=1}^{m} U_{i,lp}}{n+1} > \frac{1}{n+m+1} \\ \frac{1}{n+m+1} & otherwise \end{cases} \quad (1)$$

with the expression of shared CPU usage is as follows:

$$U_{i,lp} = \frac{\hat{U}_{i,lp}}{1+\sum_{j=1}^{m} \hat{U}_{j,lp}} + \frac{\hat{U}_{i,lp}^2}{m+1} \cdot \sum_{j=1, j\neq i}^{m} \left(1 - \hat{U}_{j,lp}\right) \quad (2)$$

These equations will provide prediction of CPU availability for each logical processor (hyper-thread) of any *c-core* processors. Here, we can see that the first part of the equation (2) computes the shared CPU usage of the I/O bound process in the run queue including its own time and time stolen from other processes. The second part of the equation computes the usage of the CPU bound process in the run queue. As this model is derived by observing a number of scheduling pattern in different and by taking the average, it can be concluded that the correlation based empirical models has been used in time series analysis for the first situation.

Instead of assigning task to a random available worker node, this model relies on a dynamic prediction monitor which is a real time monitoring utility responsible for periodic measurements of states and available resources for the worker nodes. The available resource data of worker nodes measured by dynamic monitor[7] can be used to calculate the CPU availability of each node while placing it into the computer pool. Figure 5 contains the algorithm for measuring the CPU availability of worker node dynamically.

```
While (CheckMate ≠ true) do
 IF (WorkerNodes > SiblingNodes of GameTree) then
  For (each worketNode in the pool) do
   Obtain the PID's of the N processes in the run-queue
   For ( i=1 to N)do
    Access the node's environment variable to obtains: UserTime, SystemTime, StartTime,
    and ProcessState
    IF (This process has been assigned to the processor in the last T) then
     IF (The process is in the monitor queue) then
      Calculate the shared CPU usage considering: UserTime_old, SystemTime_old, StartTime_old
      IF (ProcessState ≠ running) then
       ProcessType=I/O-bound
      Else
       ProcessType=CPU-bound
     Else
      Calculate the shared CPU usage excluding: UserTime_old, SystemTime_old, StartTime_old
      IF (ProcessState ≠ running) then
       ProcessType=I/O-bound
      Else
       ProcessType=CPU-bound
    IF (ProcessType=I/O-bound) then
     I/O-Count = I/O-Count+1
    Else
     CPU-Count=CPU-Count+1
    Update the monitorQueue: UserTime_old, SystemTime_old, StartTime_old, and ProcessType_old
  WNPool ← Evaluate the dynamic CPU availability with I/O-Count, CPU-Count, and shared
           CPU usage for that process.
  Sort the worker node pool in descending order based on processor availability results.
  Wait until the next set of tasks are available in the queue.
End
```

Figure 5: Algorithm for CPU availability prediction in distributed environment.

When new tasks are available, they are assigned to worker nodes waiting in the computer pool. Computer pool contains all available workers which has finished their assigned task and waiting for a new task. The worker nodes in the pool are arranged in descending order based on CPU availability percentage. Suppose there are three tasks waiting to be searched in distributed fashion. There are four computers waiting in the computer pool with CPU availability of 97%, 93%,

89%, and 62%. The first worker node which has CPU availability of 97% gets the highest priority as it can process the assigned task faster. We can pick tasks randomly but the worker node is selected based on the CPU availability. If the number of task is more than the available worker nodes in the pool, then tasks can be assigned randomly to any worker node. Figure 6 shows a typical task assignment procedure in which three tasks are assigned to first three waiting nodes in the pool.

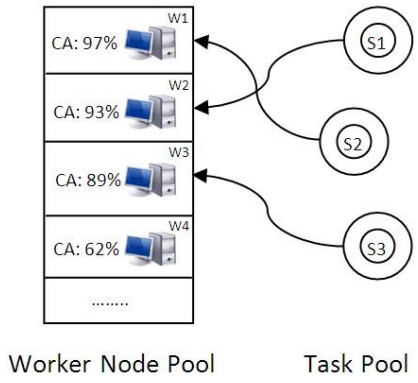


Figure 6: Task assignment to worker nodes based on CPU availability.

In games such as chess, there is a game tree consists of all possible moves by both players. Depending on the resulting board position, we can search this tree to find an effective playing strategy. Game playing is a multi-agent environment where each agent's search must try to put itself in a position to win while also preventing the opponent from putting it in a position where it is able to win. This kind of search between opposing agents is called adversarial search.

The chess game is deterministic, turn-taking, two-player zero-sum games of perfect information. So, agents have complete knowledge of the game state and the scores at the end of the game are opposite and sum to zero. As agents in games generally have a limited amount of time to make a move, so efficiency in finding a good move is important. Pruning the search tree to reduce search time, thus becomes very important.

Figure 7 shows the flow chart of the chess game of two players (between human and computer). After a human move, it always checks whether the move is legitimate move or not. If move is legal then it will calculate new move and change the board position. If it's a checkmate then the game is over else it will continue calculating new moves until one player gets checkmate and the game is over. The system will display the check mate and terminate.

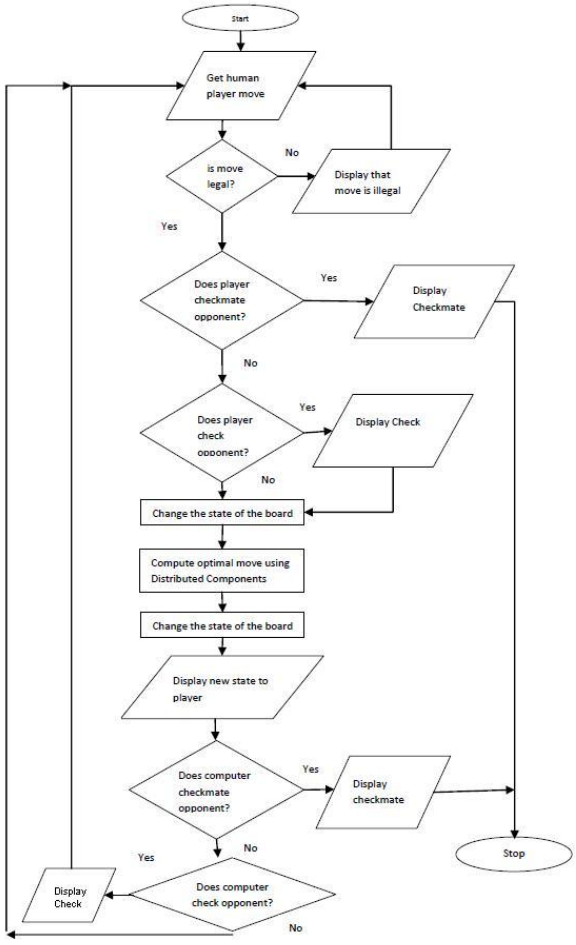


Figure 7: Program flow control depending on board position.

Figure 8 shows the GUI of the chess game board implemented during this project. This chess board can

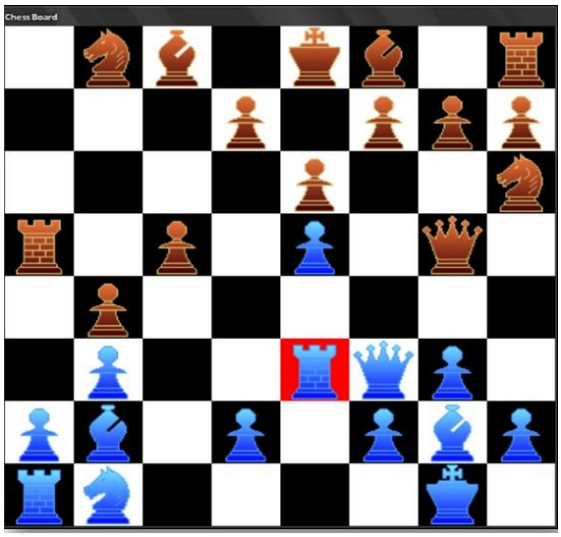


Figure 8: An instance of the implemented Chess game board position.

be operated using keyboard or mouse or both. After each legal move, the system calculates new move and updates the board pieces based on the move. It can also identify any illegal move provided by user and prompts accordingly.

The next section contains the results of several empirical studies for measuring the performance of the chess game implemented for this project in a single computer and distributed environment.

5 Analysis of Experiment Results

As the objective of the project was to create a distributed program that would reduce the time necessary to calculate moves, the system was evaluated for the amount of time taken by the program in two different scenarios. In the first scenario, one computer was used to calculate the moves. In the second scenario, three computers were used to calculate the same move.

First a set of states from which to perform the evaluations were created. Each state is a certain configuration of pieces on the board. For each of the states, move was initiated by the human player and then the time taken for the system to respond with its move in both scenarios was determined. For the first scenario a move was made and then the time taken system to make its move was observed. In the second scenario, the same move was made from the same state. The time required to make the move as well as the number of nodes searched by each computer was measured.

In the first scenario, a single pc was used and performed moves for certain states. These states are labeled 1, 9, 15, and 24. In the second scenario, the same states 1, 9, 15, and 24 were used again. In all states a remarkable decrease in the times required to calculate a move was seen. In state 1, the time decreased from 35.48 seconds to 12.5 seconds. In state 9, the time decreased from 45.07 seconds to 20.29 seconds. In state 15, time decreased from 69.87 seconds to 26 seconds. And in state 24, the time decreased from 40.06 seconds to 16.25 seconds. The results are shown as a bar graph in Figure 9.

As part of node searching, the average search speed on one PC in the first scenario was 421,946 nodes/sec. In the second scenario, involving three computers, the search speed was always over 1,200,000 nodes/sec. Thus the search speed was increased over three times by using three computers instead of one. It should be noted that the computer resources used were not homogenous. In the benefit of load balancing, the faster computers got more nodes to search. It has been observed that the dual core machine has searched the most nodes because it was the fastest.

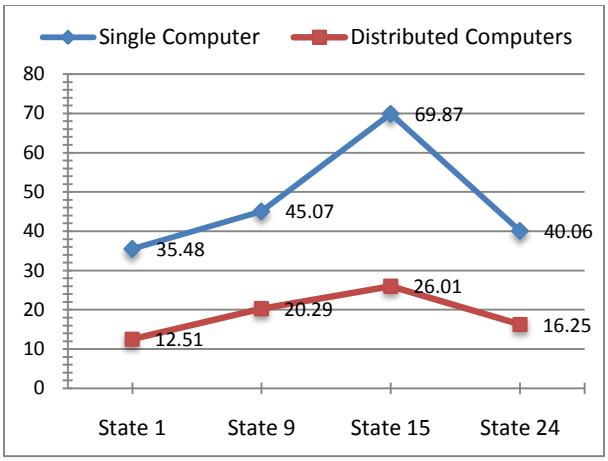


Figure 9: Comparison of time taken (in second) by the distributed system and a single computer.

The empirical results shows that the number of nodes searched by a single computer is always less than the number of nodes searched in distributed environment. This difference can reach as high as 18,708,420 nodes. Searching more nodes always helps to find better optimal solution. Therefore, this distributed system can provide much more competitive chess game than an individual computer. Figure 10 shows a comparison of the number of nodes searched between single computer and distributed system.

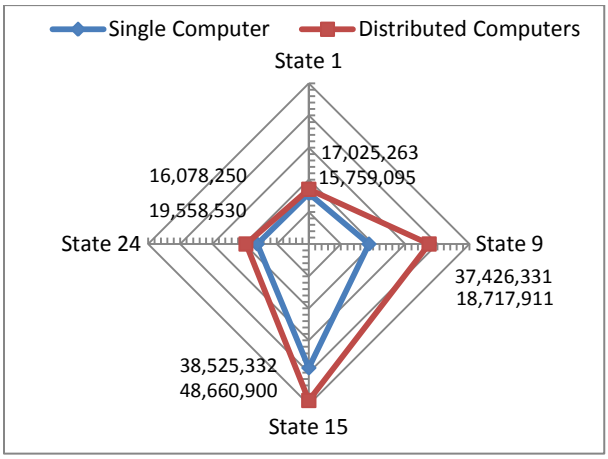


Figure 10: Comparison of total number of nodes searched by the distributed system and a single computer.

It should be noted as well, that the computer resources used were not homogenous. The different computers have different computational capabilities. PC#1 was an Athlon XP 2.4GHz, PC#2 was a Pentium 4 and PC#2 was a Core Duo 1.6GHz. In the interest of load balancing, the faster computers got more nodes to search. It can be seen from the table below that the dual

core PC#3 searched the most nodes because it was the fastest (summing nodes searched over the two cores).

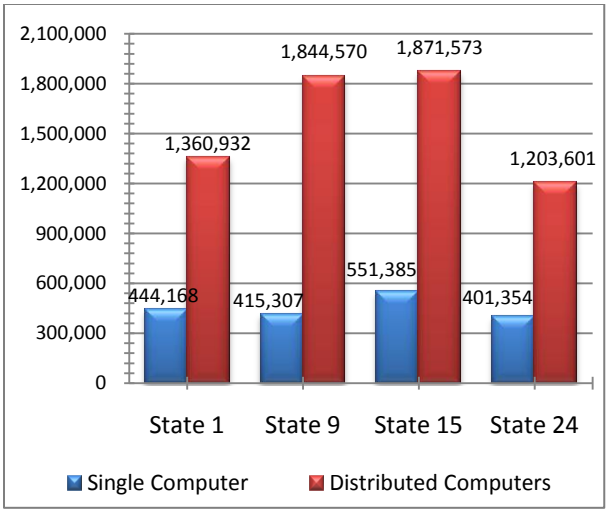


Figure 11: Comparison of search rate (nodes/ second) between a single computer and distributed system.

The rate at which nodes wear search was also calculated. The results are shown as a bar graph in Figure 11. The average search speed on one PC in the first scenario was 42,1946 nodes/sec. In the second scenario, involving three computers, the search speed was always over 1,200,000 nodes/sec. Thus the search speed was increased over three times by using three computers instead of one. (It should be noted that one of the computers was a dual core and both cores were used).

6 Conclusion

This paper has developed an analytical model (and conducted empirical studies) for predicting the CPU availability of worker nodes (single- and multi-core processors) for enhancing the performance of a distributed chess software system. This paper is a distributed computing effort in the field of artificial intelligence. The PV-Splitting algorithm modified for distributed systems was implemented and was validated. The results of the empirical study were quite impressive. The optimal moves made by the computer in response to human player moves takes far less time than while the game was running in a single computer.

The empirical studies conducted in this paper shows that the new system significantly reduces the optimal move calculation time when PV-Splitting has been employed along with dynamic CPU availability predictions. The system was tested in a case of one computer vs. three computers. The results show a large increase in speed, where speed was calculated as the number of nodes searched per second. Thus the amount of time taken to calculate a move was decreased, which was the main objective of the system.

As part of future work, we can focus on few deficiencies of the system like it cannot take advantage of more worker nodes than the number of branches of a node. The average branching factor is 35. This is a flaw of the PV-Splitting algorithm. The minimax implementation in the worker nodes can also be improved using Negascout algorithm (a directional search algorithm which is faster than alpha-beta pruning) to provide faster performance.

6 Reference

[1] Burkhard Monien, Thomas Ottmann, Data Structure and Efficient Algorithms, Springer Press, 1992.

[2] Stuart Russell, Peter Norvig, Artificial Intelligence: A Modern Approach, Amazon Press, 2008.

[3] Valavan Manohararajah, "Parallel alpha-beta search on shared memory multiprocessors", Master's thesis, Graduate Department, of Electrical and Computer Engineering University of Toronto, Canada, 2001. http://www.valavan.net/mthesis.pdf

[4] Khondker S. Hasan, Alok Chowdhury, Asif Mahbub, Ahammed Hossain, Abul L. Haque, "Implementation of a Distributed Chess Playing Software System Using Principal Variation Splitting", *16th International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, July 2010, Nevada, USA.

[5] Yaoqing Gao, T. A. Marsland, "Multithreaded Pruned Tree Search in Distributed System" http://www.cs.ualbetra.ca/~tony/Recent Papers/icci.pdf

[6] Dingzhu Du, Panos M. Pardalos, Minmax and Applications, Amazon Press, 1995.

[7] Martha Beltrán, Antonio Guzmán and Jose Luis Bosque, "A new CPU Availability Prediction Model for Time-Shared Systems", *IEEE Transactions on Computers*, Vol 57, No. 7, pp. 865-875, ISSN: 0018-9340, July 2008.

[8] R. Wolski, N. Spring, and J. Hayes, "Predicting the CPU Availability of Time-Shared Unix Systems on the Computational Grid," *Proc. Eighth International Symposium on High Performance Distributed Computing*, pp. 105-112, August 2002.

[9] *Operating System Concepts*, 8th Edition, Silberschatz, Galvin, Gagne, Wiley & Sons Inc., ISBN: 978-470-12872-5, July 2009.

EYEVISION: An Innovative Framework for the Development of Artificial Vision Systems

Maura Pasquotti[1], **Marco Del Pin** [2]
[1] Area Science Park, Trieste, Italy
[2] EIDON KAIRES S.r.l., Trieste, Italy

Abstract - *This paper contains the description of EYEVision, a framework developed by EIDON for computer vision based systems. This software is designed to be multiplatform; its modular structure allows to use processing modules from different manufacturers and its intrinsic structure reduces the learning curve for newly introduced development team members.*

Keywords: Framework, Artificial Vision Software.

1 Introduction

EIDON, which is a privately held research company, has been working for the past 30 years in the field of high performances artificial vision based systems, which comprises Real Time elaboration of high dimension images, systems with cluster of cameras and tailored elaboration to very precise cases of study that cannot be traced back to standard solutions. All the research projects developed required an initial phase for problem formulation and a following one for the development of the solution, which comprises both the theoretical solution and the software implementation.

This approach allows to obtain a solution really tailored to the specific problem but requires a lot of human resources to be applied. In particular the development of a specific new software solution each time, means that a brand new code needs to be developed for every single project, with a lot of time wasted and a requirement of knowledge of the whole project by the software developer to effectively realize the software.

The objective is then to develop a configurable artificial vision software that allows the researchers to generate a complete system without starting from scratch, but re-using the knowledge acquired from the previous projects and adding only the specific algorithms required to solve the new problem.

At the state of the art, in both industrial and academic fields, there are some software tools to help developers to reuse code in order to create a new implementation. Some of these solution are well established and cover a wide range of standard applications (for example smart cameras, see [8] for further details) while others are specifically developed for a narrow field of interest (for example [9] a vision platform for mobile robot applications) and are very specialized.

However none of them presents both the characteristics of flexibility and high performances which are required to develop complex computer vision systems that operate in completely different environments, as Eidon currently does.

To overcome these limitations a brand new platform has been developed in the previous years and EYEVision is the results of this research line.

As analogy in the building industry, old development scheme can be considered as a house built brick by brick while the EYEVision way can be represented as a luxury prefabricated house, a quicker and cheaper version of the previous construction method.

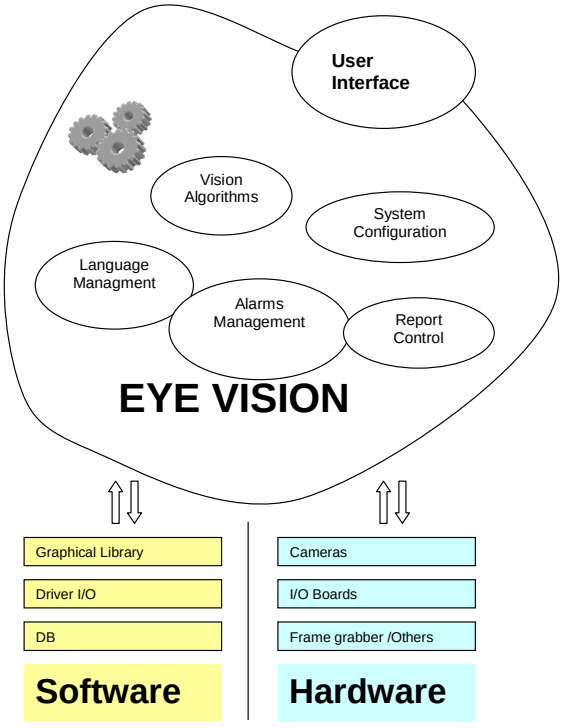


Figure 1. EYEVision Structure

2 Architecture

The EYEVision software has been designed with re-using patterns. The whole software can be divided in three different parts:

- The Acquisition Unit;
- The Elaboration Unit;
- The Display Unit.

Each part is independent from the others and can be developed separately.

The innovative idea implemented in the software is that each module communicates with the rest through an abstract interface layer, which is independent form the actual implementation, and is a plug-in style software

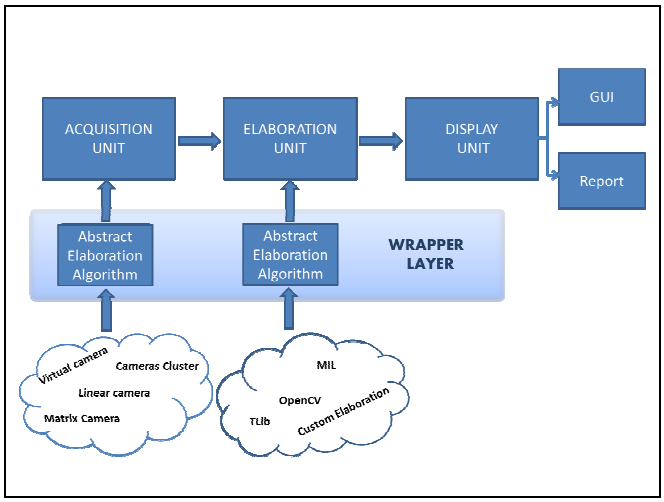


Figure 2. EYEVision Architecture.

.

3 Acquisition Unit

The idea that lays under the structure of the acquisition module is the presence in the software of a "generic camera" object, that interacts with the rest of the software..

This generic object is an abstract interface, under which there is the real device used. This scheme guarantee that changing the input device means that a new wrapper needs to be implemented, while the rest of the software remains untouched.

In this way the input camera can be easily changed and for example a virtual camera, that reads the data stored in the HD instead of the frames grabbed, can be realized.

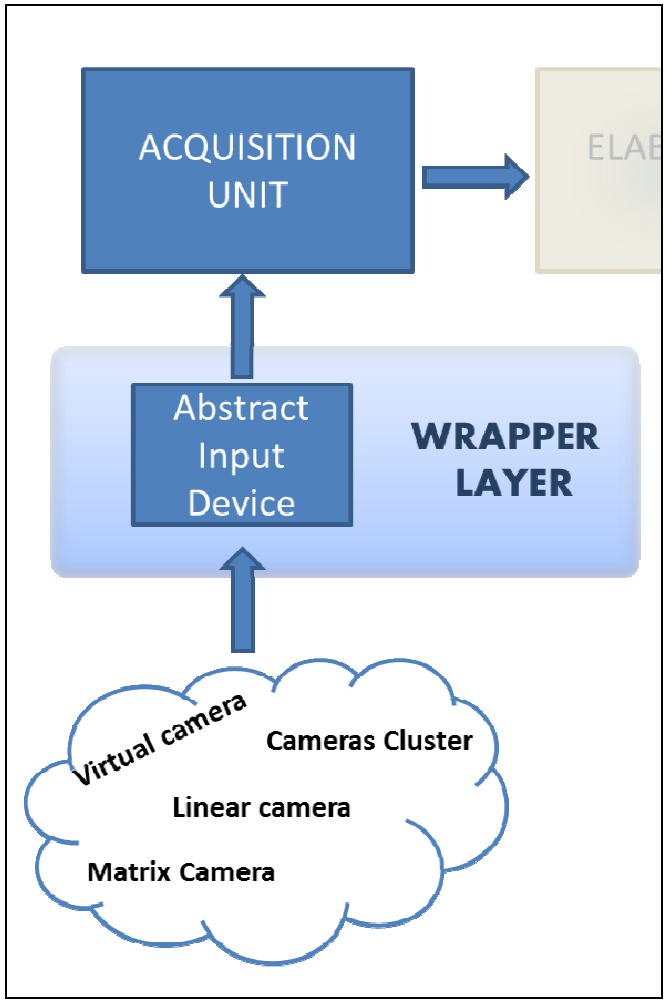


Figure 3. Acquisition Unit.

4 Elaboration Unit

Also the elaboration part of the software shows a plug-in structure. In this part there are a catalog of well-established image elaboration algorithms that can be enhanced with ad-hoc elaboration.

To add a new elaboration to the system the programmer need to follow some rules, but doesn't need to write all the code from scratch.

The presence of the wrapper layer in the software allows the developer to choose the appropriate graphical library that mostly fits their elaboration requirement without worrying about rewrite all the standard elaborations. In fact all the elaborations are one of this:

- Custom developed processing that requires innovative elaboration;
- Image processing that relies on standard algorithms.

In the latter case the standard algorithms are not directly called by the function, but are called through a library wrapper; changing the library, the elaborations remain the same and the only part that needs to be rewritten is the wrapper interface for the new library added. In particular the graphic library could be chosen between several options, that includes the OpenCV (Open Source Computer Vision library [3]), the TLIB ([4]), the MIL (Matrox Imaging Library [2]).

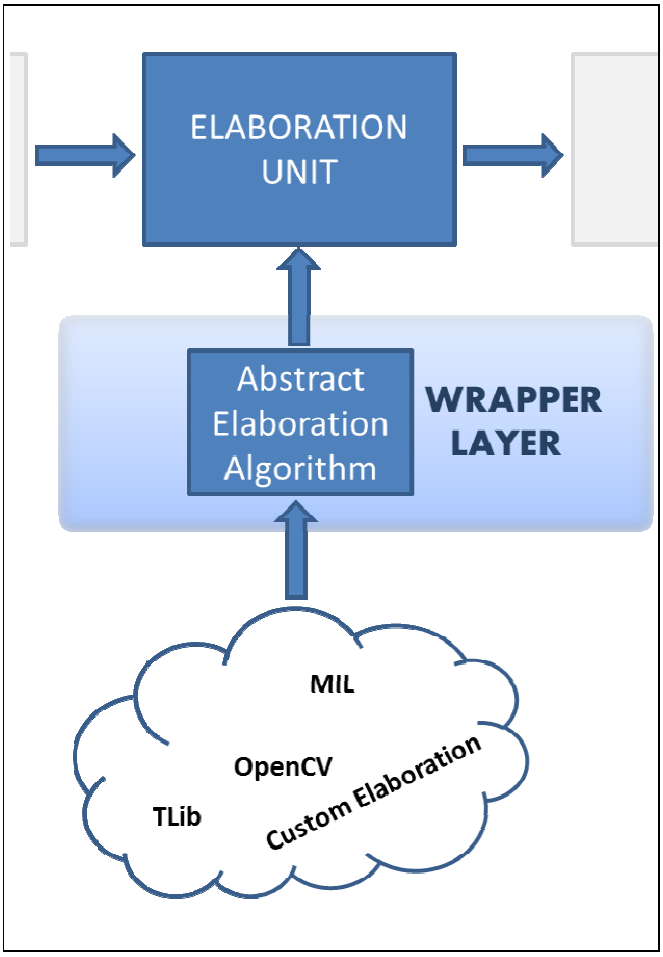


Figure 4. Elaboration Unit.

5 Display Unit

The Display Unit includes all the visual interaction between the user and the software.

It is composed by a Graphical User Interface (GUI) which shows on the monitor the results of the elaborations and the system status for monitoring purposes. The GUI presents some panels to calibrate the system before its normal functioning and in case also to define a new set of rules to create a different elaboration.

As well as the other parts of EYEVision, also the GUI presents a modular structure and is easily configurable; there is the possibility of adding various panels, starting from simple standard winform to more advanced vector graphic representation. There is also the possibility of adding data through the network.

In addition there is also a report module for statistics purposes, which is capable to store in a database all the results of tests carried out, and to retrieve them in order to see the analysis and their trends, to print reports that display the results obtained from the system analyzed in terms of pieces per lot, per shift, etc..

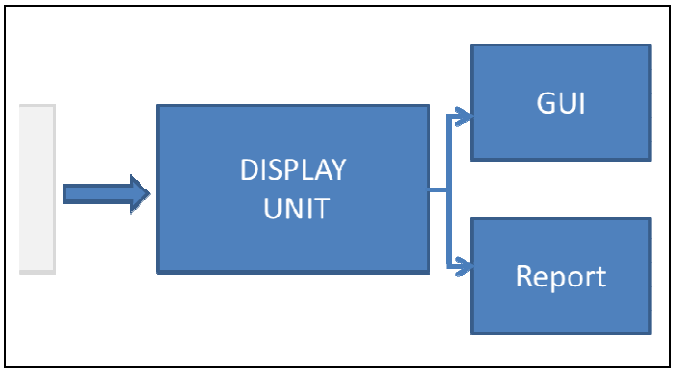


Figure 5. Display Unit.

6 Cases of Study

There are several examples of systems that can be easily generated with the EYEVision framework; figure 2 shows some possibilities.

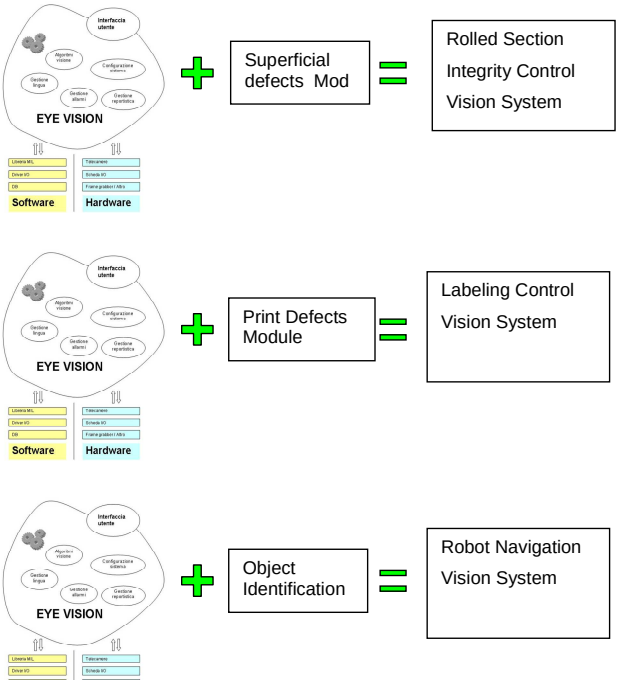


Figure 6. Example of system implementation with EYE Vision

In this paper we focus our attention to the specific problem of acquisition, classification and registration of natural stone slabs.

The physical system designed to solve the problem is essentially composed by a linear color camera and a conveyer belt for slab handling, as shown in figure 7.

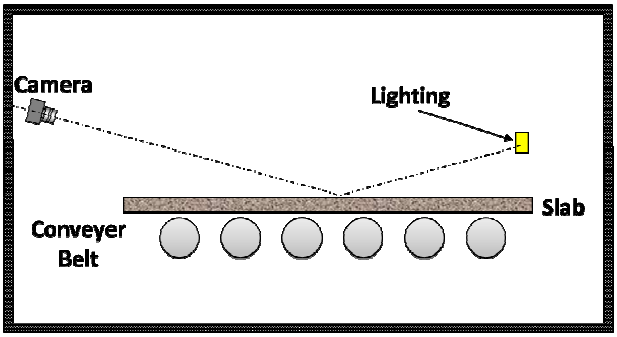


Figure 7. Slab Acquisition System.

Another requirement of the problem is the capability of the final system to supervise several acquisition units physically located in different places.

The analysis developed need to solve the following problems:

- Slab profile detection;
- Maximum usable slab surface;
- Slab pattern homogeneity recognition;
- Slab color homogeneity recognition;
- Slab quality calculation.

Some of the analysis required for this specific problem case can be carried out by standard algorithms, while others require ad-hoc solution.

The software development consists in the coding of the following:

- A wrapper for the linear color camera;
- A custom algorithm for the calculation of the maximum rectangle inscribed in the slab, in order to determine the biggest regular piece that can be cut off the slab;
- A custom algorithm for inhomogeneity detection for color and surface patterns;
- A module for multiple systems aggregation and monitoring.

All the rest of the code required to successfully solve the problem (acquisition management, interaction between acquisition devices and elaboration, graphical results presentation, etc.) is already a part of EYEVision software, so doesn't need to be implemented again (in figure 8 there is an example of a standard GUI panel for the slab detection project).

All the custom modules development can be carried out by a single person or split among a development team; in both cases each person doesn't need a deeply knowledge of the details of the whole system, but needs only to know the communication interfaces and the common rules for module implementation. Moreover, since EYEVision platform is realized in .NET development environment, a new module can be virtually written using any of the several programming languages supported.

This means that a new team member doesn't require a long and intense training period before effectively giving their contribution to the development process.

The time saved using this approach instead of developing the solution from scratch is considerable.

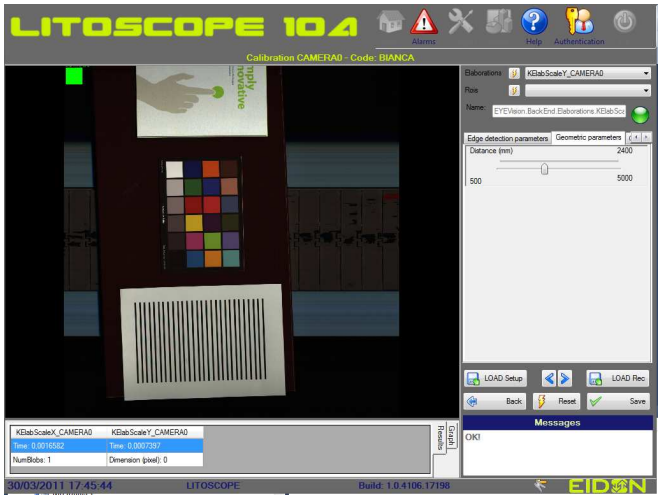


Figure 8. Graphical User Interface.

7 Conclusions

Current resource and time-to-market constraints on the software development process push developers to create a final product as quickly as possible. To accomplish this task developers need to base their development on successful past experiences [1]. Moreover, they should be able to reuse these experiences in well-structured computer-aided processes. In this context, the EYEVision framework appears to be a promising approach.

In this paper we have described its structure and pointed out the advantages of this framework, which includes:

- Reusability. The component is not recoded when the components are used in other domains or contexts, because the component implementation can be adapted to new business rules by changing the non-intrinsic dependencies. Then these components can be coupled with others components
- Adaptability. Programmers are offered the possibility of modifying the component descriptor by altering the final component functionality.
- Scalability. The system can be easily scalable because we obtain new component implementations and new component specifications. Then these new components with their dependencies can be used to compose new systems.
- Compressibility. Developing a new system is based on following a set of structured phases (Design and Specification, Implementation, Package, Assembly and Deployment).
- Reliability. The reuse of components already tested and approved increases the safety of the program

Acknowledgment

The authors would like to thank Mr Santoro and all the EIDON team for their support, and the AREA Science Park, Trieste, Italy.

References

[1] G Bertrand, M., Object-Oriented Software Construction, New York, NY: Prentice Hall, 1988.

[2] http://www.matrox.com/imaging/en/products/software/

[3] http://opencv.willowgarage.com/wiki/

[4] Sebastien Grange, Terrence W. Fong, and Charles Baur, "TLIB: A real-time computer vision library for HCI applications," Digital Image Computing - Techniques and Applications Conference, December, 2003.

[5] Dijkstra, Edsger W. A Discipline of Programming. Englandwood Cliff, NJ: Prentice-Hall, 1976.

[6] Fayad, M. E., M. Cline. Aspect of Software Adaptability. Communications of ACM, Vol. 39, No. 10, pp.58-59, 1996

[7] http://nuke.eidon.it/

[8] http://en.wikipedia.org/wiki/Smart_camera

[9] Sandor Szabo, David Coombs, Martin Herman, Ted Camus and Hongche Liu, “A Real-time Computer Vision Platform for Mobile Robot Applications”, Real-Time Imaging Volume 2, Issue 5, October 1996, Pages 315-327

A Petrinet Realization of an Urban Traffic System

Kuhelee Roy[1], **Ranjan Dasgupta**[2]

[1] Dept. of Computer Sc. & Engg , National Institute of Technical Teachers' Training & Research , Kolkata, West Bengal, India(student)

[2] Dept. of Computer Sc. & Engg , National Institute of Technical Teachers' Training & Research , Kolkata, West Bengal , India

Abstract - *The requirements of system reflect the needs of the customer. Specification of requirements at different levels of abstractions is therefore an important part of system design. Formalism in part of system design is necessary to avoid ambiguity and provide precision. Petrinet is a design tool which has proven to be very promising in this aspect. This paper deals with the application of petrinet in control of pedestrian signals in conjunction with vehicle signals in an urban traffic system. The petrinet realization is followed by defining some constraints which is indispensable in enforcing safety with regard to pedestrians as well as vehicles involved in the traffic system.*

Keywords: Petrinet, Safeness Constraints, transition, marking, token

1 Introduction

Modeling is beneficial for a number of reasons like verifying the correctness of the system, detecting the presence of deadlocks, discovering errors, measuring system performance etc. Hence modeling of complex systems provides a research field that spans a wide area. A wide range of modeling tools has also been devised for this purpose. Petrinet has proven to be one such promising modeling tool in designing complex systems. Today the increasing demand of better control measures in traffic management system demands a more efficient modeling in terms of vehicle and pedestrian control. This paper deals with the application of petrinet in control of pedestrian signals in conjunction with vehicle signals in an urban traffic system followed by defining some constraints which is indispensable in enforcing safety with regard to pedestrians as well as vehicles involved in the traffic system.

2 PETRINET

Petri Net, developed in the early 1960s by Carl Adam Petri, are graphical and mathematical modeling tool for describing and studying systems that are concurrent, asynchronous, distributed, parallel, non-deterministic[1].

Petri nets are defined by a quadruple [11] (P, T, F, W), where

1. P is a finite set of places, represented by circle or ovals.

2. T is a finite set of transitions, represented by squares or rectangles.

3. $F \subseteq \{P \times T\} \cup \{T \times P\}$ is the flow relation, represented by arcs. Arcs connect places and transitions. Inward arcs go from a place to a transition and an outward arc goes from a transition to a place.

4. $W: F \rightarrow N-\{0\}$ is the weight function, which associates a nonzero natural value to each element of F. If no weight value is explicitly associated with a flow element, the default value 1 is assumed for the function.

Tokens reside in the places. They are represented by a solid circle or by a dot inside of a place. The execution of Petri net is controlled by the position and movement of tokens. A Petri net is a kind of directed graph with intial marking M_0. A marking (state) assigns to each place a positive integer. If a marking assigns a non-negative integer k to a place p, then p is said to be marked with k tokens. A marking is denoted by M, an m-vector. The pth component is denoted by M(p), which represents the number of tokens at place p[1]. For a given marking M the firing of an enabled transition (transition becomes enabled on presence of tokens in the input places of the transition) results in a successor marking M',

Where, $M'(p) = M(p) - w(p, t) + w(t, p)$,

Where, w(p, t) is the weight of the arc between p and t.

w(t, p) is the weight of the arc between t and p.

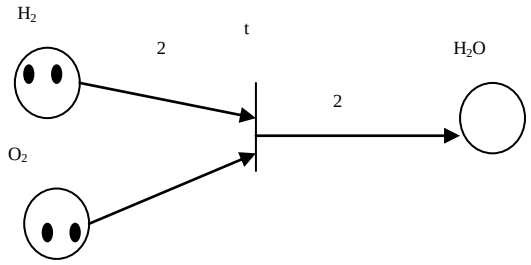


Fig 1. Initial marking

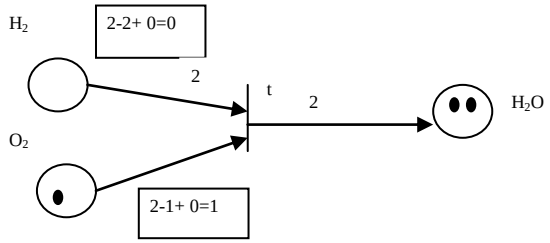


Fig.2. After firing of t

A drawback of petrinet is that all tokens are identical, that is, there is no way to associate any additional data with token. To tackle this issue, colored petri nets are used. With Coloured Petri Nets (CP-nets) it is possible to use data types and complex data manipulation. Each token has attached a data value called the token colour. The token colours can be modified by the occurring transitions[2].

3 Applications of petrinet

Petrinet has its application in various fields like communication systems, synchronization control, producer-consumer system, formal Languages, multiprocessor systems[1]. Besides, the applications of petrinet are many-fold. Petrinet has been used in modeling the control at signalized intersections for traffic movements and / or phases(movements with simultaneous greens)[3] thereby accounting for indication display module for a single movement and phase transition between phases. The modeling power of petrinet has been utilized to support the analysis of the system performance during the charging function of a GPRS network[4], followed by calculation of the maximum BHCA (busy hour call attempt) per hour. Petrinet has also proved to be a very efficient design tool in modeling railway networks comprising of track, points, station, n-tracks, followed by definition of GMECs (generalized mutual exclusion constraints) for enforcing safety[5]. Besides, a modified form of petrinet has been used in terms of adding operators like AND, OR, EX-OR etc in order to describe the control and data flow in a distributed system[6]. A model incorporating the priority/preemption request of transit vehicles has been modeled using CPN[7] which provides a priority evaluation procedure to provide different priorities to multiple priority requests from transit vehicle and emergency vehicles. Object Oriented Petri Nets provides a design methodology that consist of petri nets in form of classes, each consisting of object net and method nets. This design tool has been used in modeling a conference system[8], in terms of author, paper etc[8]. A conditional colored Petri net (CCPN) approach has been proposed to model ECA rules of an Active database [9]. A concept of APN(Advanced petrinet) has been proposed in modeling and analysis of a MAC Protocol used in Token Ring Networks [10].

However, designs for traffic signals in an urban area must consider the needs of pedestrians as well as vehicles. Application of petrinet in designing traffic control signal in light of vehicle control has already been presented in [3]. Furthermore, petrinet can be applied to control pedestrian movements in conjunction with vehicle movements.

4 Proposed model for pedestrian control and vehicle control

4.1 Pedestrian signal control for a single lane

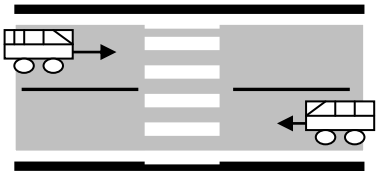


Fig.3. A single lane

For a single lane, there can be either leftward or rightward vehicles. When pedestrians are allowed to cross the road, both the leftward and rightward vehicles need to be stopped.

TABLE 1 : NOTATIONS USED (IN PN)

1. VR	Place	Red signal for leftward and rightward vehicles
2. PGF	Place	Pedestrian Green Flashing
3. V_RG	Place	Green for rightward vehicle
4. V_LG	Place	Green for leftward vehicle
5. PR	Place	Pedestrian Red
6. Y	Place	Yellow
7. V_RR	Place	Red for rightward vehicle
8. V_LR	Place	Red for leftward vehicle
9. PG	Place	Pedestrian Green
10. CNTDWNTIMER	Place	Countdown timer to alert the pedestrians of the time remaining for safe crosswalk
11. PCW	Place	Pedestrians on crosswalk
12. PEG	Place	Pedestrian extended green
13. FO	Transition	Force-off(The current phase is terminated)
14. PS-T-D	Transition	Pedestrian Speed and Time detection.

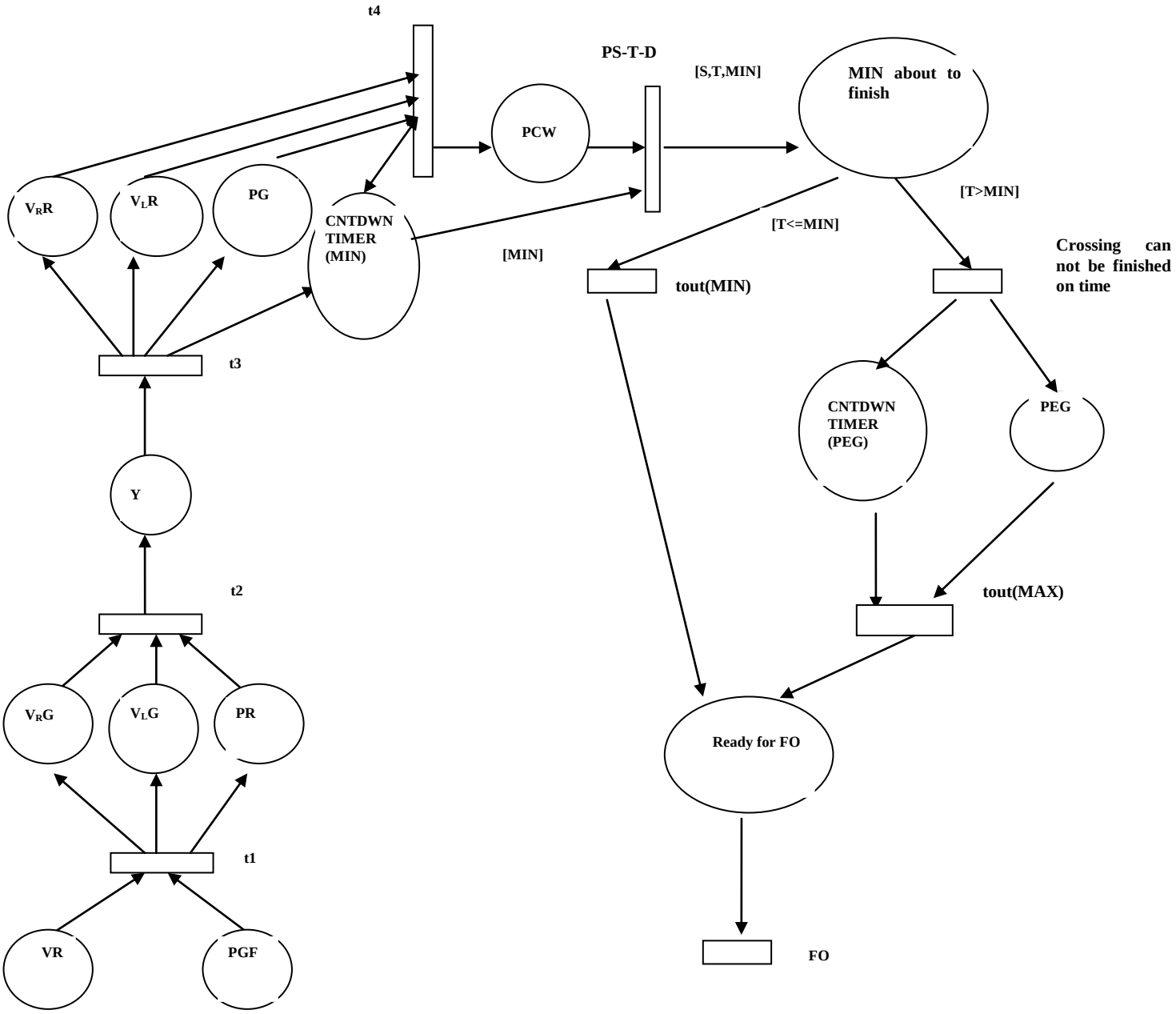


Fig.4. Petrinet for pedestrian crossing on a single lane

4.2 Petrinet Realization of pedestrian crossing a single lane

Initially, the presence of tokens in VR and PGF denotes that the current phase shows a) red for both rightward and leftward vehicles. b) green flashing for pedestrians .Firing of transition t1 deposits tokens in places VRG and VLG(green is being shown for rightward and leftward vehicles) and PR (red is shown for the pedestrians). When transition t2 becomes enabled, it fires and deposits token in the place Y(Yellow). Transition t3 is enabled on presence of token in Y(yellow) and hence fired thereby depositing tokens in the places V_RR, V_LR (red is shown for rightward and leftward vehicles), PG (pedestrians can start crossing) and CNTDWNTIMER(MIN)(a timer is started when the

pedestrians start crossing, min denotes the minimum stipulated time for safe crosswalk of the pedestrians). Transition t4 becomes enabled on presence of tokens in the places V_RR, V_LR, PG and CNTDWNTIMER(MIN). A double-headed arc has been shown between CNTDWNTIMER(MIN) and transition t4. This denotes that after t4 is fired a token is returned to CNTDWNTIMER(MIN), thereby asserting that the countdowntimer for the minimum time required for safe crosswalk is still on while pedestrians are crossing the road. Firing of transition t4 also deposits token in the place PCW. When pedestrians are on crosswalk it is required to detect the average speed of the pedestrians in order to check if the crosswalk can be completed within the minimum stipulated time for safe crosswalk. This is denoted by firing of the transition PS-T-D. This transition has two significances:

a. The average speed of the pedestrians on the crosswalk is determined. Therefore there is an input arc from the place PCW to the transition PS-T-D.

b. The expected time required for the pedestrians to complete the crossing is also determined which needs to be compared with the minimum stipulated time for safe crosswalk. Therefore an input arc is present between the place CNTDWNTIMER(MIN) and the transition PS-T-D. This input arc is labeled with [MIN] which denotes that the minimum time required for safe crosswalk is provided for the comparison.

Firing of the transition PS-T-D deposits token in the place MIN about to finish. The output arc of the transition PS-T-D is labeled with [S,T,MIN] , where S stands for the average speed of the pedestrians on the crosswalk, T stands for the expected time for the pedestrians to complete crossing, MIN is the minimum stipulated time as provided from CNTDWNTIMER(MIN). At this point two conditions can arise which is denoted by the presence of two outgoing arcs from the place MIN about to finish:

One of these arcs is labeled with [T<=MIN], that is, if the expected time for crosswalk is less than or equal to the MIN. This arc acts as an input arc for the transition tout(MIN). Firing of transition tout(MIN) denotes the minimum time for safe crosswalk is over.

The other arc is labeled with [T>MIN] which denotes the case when pedestrians already on the crosswalk(PCW) can't finish crossing within the stipulated time. This is represented by firing of the transition Crossing can not be finished on time. At this point there is a need for extended green(PEG) for the pedestrians, thereby depositing tokens in the places PEG. At the same time another timer for the PEG is also started which is represented by deposition of tokens in the place CNTDWNTIMER(PEG).

Finally the maximum green for the pedestrians is over (transition tout(MAX) fires). If situation demands even more pedestrian green then a force-off is required, where the phase of pedestrian green is terminated. The need for force-off can arise on two conditions:

a) when MIN is over. This is denoted by presence of arc from tout(MIN) to the place Ready for FO.

b) when MAX is over. This is denoted by presence of arc from tout(MAX) to the place Ready for FO.

4.3 Safeness Constraints

These constraints refer to the constraints on the marking of the places of the petrinet at a given time. These constraints enforces safety in the urban traffic system. Hence these constraints are referred to as safeness constraints.

1. PGF +VR = 2

This constraint states that, at any given time, the sum of tokens at PGF and VR should be equal to two. Both the places should have tokens simultaneously. This asserts that pedestrian green flashing and red for vehicles should exist simultaneously.

2. $PR + V_RG + V_LG = 3$

This constraint asserts that places *PR,* V_RG *and* V_LG should have tokens simultaneously. This asserts that red for pedestrian and green for rightward and leftward vehicles should exist simultaneously.

3. $PG + V_LR + V_RR = 3$

This constraint asserts that places *PG,* V_LR *and* V_RR should have tokens simultaneously. This asserts that green for pedestrian and red for rightward and leftward vehicles should exist simultaneously.

4. *(PEG + CNTDWNTIMER(PEG)) + (PG + CNTDWNTIMER(MIN)) = 2*

This constraint asserts that the system can be either in a state of min green for pedestrians or extended green for pedestrians. In fact, extended green is reached when the system has already been through min green for the pedestrians.

5 Conclusion

The following problems regarding the control of pedestrian movements in conjunction with the vehicle movements has been addressed in this paper:

1. When it is detected that pedestrians cannot finish crossing before the light turns to red, the green time for pedestrians is extended for a few more seconds.

2. A measure is to present to alert pedestrians regarding the remaining time available for safe crosswalk.

3. A force-off needs to be enforced under certain conditions.

The need for integrating the control for pedestrian movement and vehicle movement in a traffic system has lead to the inception of modeling the urban traffic system using petrinet in the light of controlling the moves of pedestrians in conjunction with vehicles. The petrinet is followed by introduction of some safeness constraints in section 4.3 that are indispensable for ensuring safety in urban traffic system in terms of pedestrians and vehicles.

6 References

[1] Petri Nets: Properties, Analysis and Application, Proceedings of the IEEE Tadao Murata ,Vol.77, No.4, April 1989

[2] K. Jensen, L.M. Kristensen, L. Wells: Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. International Journal on Software Tools for Technology Transfer, (2007), Springer Verlag, 213-254.

[3] George F. List, Mecit Cetin, "Modeling Traffic Signal Control Using Petri Nets", IEEE Transactions on Intelligent Transportation Systems, Vol. 5, No.3 , September 2004,

[4] Jiacun Wang ,"Charging Information Collection Modeling and Analysis of GPRS Networks", IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and reviews, vol.37, No.4, July 2007

[5] Alessandro Giua, Carla Seatzu," Modeling and Supervisory Control of Railway Networks Using Petri Nets", IEEE Transactions on automation science and engineering, Vol.5, NO.3, July 2008

[6] Stephen S. Yau, Mehmet U. Cagalayan, "Distributed Software System Design Representation Using Modified Petri Nets", IEEE Transactions on Software Engineering, Vol. SE-9,NO.6,November 1983

[7] Lefei Li, Wei-Hua Lin, and Hongchao Liu, " Traffic Signal Priority/Preemption Control with Colored Petri Nets", Proceedings of the 8th International IEEE Conference on Intelligent Transportation Systems Vienna, Austria, September 13-16, 2005

[8] Radek Koˇc´ı, Vladimir Janouˇsek, and Frantiˇsek Zboˇril, jr.,"Object Oriented Petri Nets – Modelling Techniques Case Study", Second UKSIM European Symposium on Computer Modeling and Simulation, 2008 IEEE

[9] Joselito M. Medina, Sergio V. Chapa, " Applying Petri Nets in Active Database Systems", IEEE Transactions on Systems, Man, and Cybernetics-Part C: Applications and reviews, vol.37, No.4, July 2007

[10] K.S. Sivanandan K. Garg N.K. Nanda "On the petrinet Modeling of Mac protocol", IEEE Region 10 Conference on Computer and Communication Systems, September 1990, Hong Kong

[11] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli, " Fundamentals of Software Engineering", Second Edition, Pearson Education, Inc.,2003

[12] J. Wang, "Petrinets for dynamic event-driven system modeling", Handbook of Dynamic System Modeling, Ed: Paul Fishwick, CRC Press, 2007

Designing an Online Conference Management System

Kevin Daimi and Luming Li
Department of Mathematics, Computer Science and Software Engineering
University of Detroit Mercy,
4001 McNichols Road, Detroit, MI 48221
{daimikj, lilu}@udmercy.edu

ABSTRACT

Academic conferences play a key role in exchanging research ideas between participants and keeping researchers current. Many academic conferences, in various fields, are held annually. This leads to a dramatic increase in the number of submitted papers, and substantial effort to manage these many submissions. Such an intricate workflow of conference management results in frustration among many conference organizers. In this paper, we propose an online system to support the organization, management, and control of academic conferences.

Keywords

Conference Management Systems, Software Requirements Engineering, Software Engineering, Database, User Interface Design

I. INTRODUCTION

For faculty and researchers, attending at least one academic conference annually in their fields of interest is inevitable. In such conferences, many stakeholders are involved in various conference tasks. These include, but are not limited to, program committee chair, program committee members (reviewers), general chair, publicity chair, and authors [3], [4], and [5]. For a conference organization to be successful, a process should be in place. The process of conference organization consists of many phases, such as call for papers, paper submission, paper review, review discussion, paper re-submission, and author notification [5]. Stakeholders with varying viewpoints, in addition to the complex conference organization process, make organizers, especially those without any prior professional organization skills, feel unenthusiastic about managing an academic conference, and possibly quit the task.

With the presence of advanced technology affecting all perspectives of our life, academic conferences are increasing in great number. This is accompanied by an enormous increase in the number of submitted papers. To cope with such large a number of papers and to keep reviewing loads manageable, the number of program committee members has to significantly increase. Consequently, scheduling a face-to-face program committee meeting to review and confer paper submissions is deemed impractical [2].

Based on what is mentioned above, it is vital to develop an online conference management system that facilitates the task of conference organization using the techniques of software engineering. Software Engineering first emerged in the 1968 NATO Software Engineering Conference [10], and [11]. It is defined as "The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software" [11]. Currently, various methods are available to assist software engineers in the analysis, specification, design, implementation, and verification of software products. Furthermore, software developers have improved their comprehension of the activities involved in the software development process. Generally speaking, the software engineering process is divided into the following sub processes: requirement engineering, software design, testing, and maintenance [6].

The first step in the software development process is requirements engineering (RE). RE defines the purpose of a software product, the constraints on it, and the needs of the stakeholders [6], and [12]. RE is generally an intricate interaction and negotiation process involving different stakeholders, such as customers, designers, managers, government and legal bodies, testers, and maintainers. The goal of requirement engineering is to extract functional requirements and non-functional requirements for the software product. In other words,

RE deals with making decisions on what needs to be done and when should it be done, the constraints that need to be satisfied, what information the proposed system should provide, and the tools used to achieve the goal [6], [13] and [16]. It should be obvious that requirement engineering is the most important sub-process. The outcomes of requirement engineering are tightly interconnected, and they direct all the subsequent subprocesses of the software development process. A number of software projects have failed due to unclear, ambiguous, inconsistent, and incomplete requirements [6], and [19]. Requirements are listed in natural languages to facilitate understanding them, and various models are used to assist in their analysis. Software engineers and researchers argue that such a combination of the list of requirements and the models used is a superior approach to improve the quality of the requirement engineering process. Modeling techniques are usually communicative, accurate, and promote the development team's specifications and grasping of the requirements. Lists of requirements expressed in natural language represent an agreement between customers and developers, and they abridge requirements management [1], [6], and [14].

The subprocess following requirement engineering is software design. Software design is the fundamental focus of software engineering. During the design phase, the software developer concludes how the software system will be constructed [6], [18], and [20]. The first step in the design is determining the software architecture, which is the set of principal design decisions [7], and [18]. Using the requirements specification, software designers employ the functional requirements mainly to construct the system's architecture [15]. However, this is not an efficient practice for designing high quality software. To achieve high quality, nonfunctional requirements, such as performance, safety, portability, security, and maintainability should be incorporated into the architecture [16]. Software architecture comprises efforst in modeling and representation, design techniques, analysis, visualization, converting the designs into code, furnishing reuse, and deployment [7], and [18].

In this paper, we present an Online Conference Management System to assist the conference chair in organizing the academic conference that follows the process of software engineering. It is a browser/server (B/S) based system, and runs under the .NET platform. The programming language C# is used to implement the system. The rest of the paper is organized as follows: in Section II, system requirements, both functional and nonfunctional, will be presented. Section III will introduce the characteristics, architecture, and user interface of the proposed system. Future work and conclusions are presented in the last section.

II. SYSTEM REQUIREMENTS

Requirement engineering is the first and most important step of software development. It states the customer's needs that the system must satisfy. The outcomes of requirement engineering ─functional and non-functional requirements─ are critical to the success of any software project [6].

A. Functional Requirements

Functional requirements state what functions the system should perform. A sample list representing the functionality of the Online Conference Management System is given below.

1. The system should allow its users (chair, reviewer and author) to sign in using their user name and password.
2. The system should allow its users (chair, reviewer and author) to upload their personal information.
3. The system should allow the chair to submit basic conference information, which includes conference name, URL, subtitles, and main organizer's contact information.
4. The system should allow the chair to set and modify the deadline for paper submission, paper review, acceptance notification, and the uploading of accepted paper (camera ready-copies).
5. The system should allow the chair to set up the topics of interest.
6. The system should allow authors to submit an abstract of a paper in addition to the paper title, authors, their emails, addresses, keywords, and the topic of interest that applies to their paper.
7. The system should allow authors to upload the full paper in a specific format (e.g. PDF file or Word file).
8. The system should allow reviewers to specify the topics that falls into his/her area of expertise.
9. The system should allow reviewers to bid for papers that they are interested in reviewing.
10. The system should allow reviewers to indicate any conflict of interest.
11. The system should be able to assign papers to reviewers automatically.
12. The system should allow a reviewer to submit an evaluation of a paper that was assigned to him/her.

13. The system should allow the reviewer to submit a special comment that can be read by other reviewers.
14. The system should allow the chair to make the final decision on accepting or rejecting a specific paper.
15. The system should allow the chair to assign an accepted paper to a specific conference session.
16. The system should allow the author to upload the camera-ready copy of an accepted paper.

B. Nonfunctional Requirements

Non-functional Requirements describe quality measures by which a software product must abide. In this section, several non-functional requirements are presented. These requirements cover the performance, security, reliability, availability and maintainability of The Online Conference Management System.

1. The system should allow every user to sign in within 5 seconds.
2. The system should be able to display the detailed conference information, within 5 seconds, when requested by chair.
3. The system should be able to list a summary of all the system users' records (authors and reviewers), when requested by the chair, within 5 seconds.
4. The system should be able to display the detailed information of a specific user, when requested by that user, within 5 seconds.
5. The system should be able to list a summary of all the paper submission records, when requested by the chair, within 20 seconds.
6. The system should be able to list detailed paper submission information, when requested by the chair, the author, or the reviewer, within 5 seconds.
7. The system must grant each authorized user a unique ID, user name, and password.
8. The system shall authenticate each user with a unique identification.
9. The system should ensure thatthe user's personal information is confidential.
10. The system should ensure that the user's identification and password cannot be modified by any other person.
11. The system shall guarantee that the paper upload and paper download activities are processed in a strictly secure manner.
12. The system shall send emails to the users in a strictly secure manner.
13. The system should be available 24 hours per day until the conclusion of the conference.
14. The system should be able to detect and correct faults.
15. The system's mean-time-between-failures should be at least 6 mouths.
16. The system should be able to restart after a failure.
17. The system should be able to back up on daily basis.
18. If the system crashes, it should be able to recover using backup copies.
19. The backup copies should be stored in an external storage device and placed in a secure location.
20. The system should be able to correct errors automatically.
21. The system should allow future improvements.
22. The system should be able to generate fault reports automatically.
23. The system should be able to generate various logs to record user's behavior.
24. The system should be able to export fault reports into files.
25. The system should be able to export logs into files.

C. Use Case Modeling

Use case modeling is a graphical representation form of functional requirements [1], [6], and [7]. In this modeling, the use case diagram consists of several possible scenarios related to the usage of the software system. In the object-oriented (OO) analysis and design process, use cases are the key input to the design phase. From a use case diagram, a developer could acquire an unambiguous idea about the system's boundary, actors involved in the system, and the actions the actors can perform. Figures 1 and 2 illustrate the use cases of the Online Conference Management System.

III. SYSTEM DESIGN

Software design is a creative task. During the design phase, a software developer concludes how to implement a software system to meet customers' needs [6], and [18]. A good design should exhibit high cohesion and low coupling [6]. Many design methods have been put forward to achieve this goal. Several case tools have matured to the point that they can help developers efficiently walk through the design process. For our design approach, a UML-based case tool, Sparx Systems' Enterprise Architect, as well as Microsoft Visio are used.

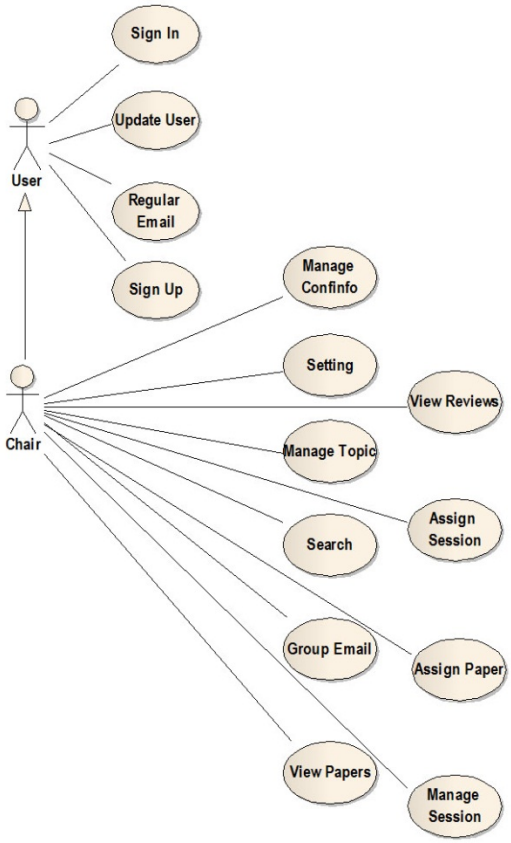


Figure 1: Use Case 1

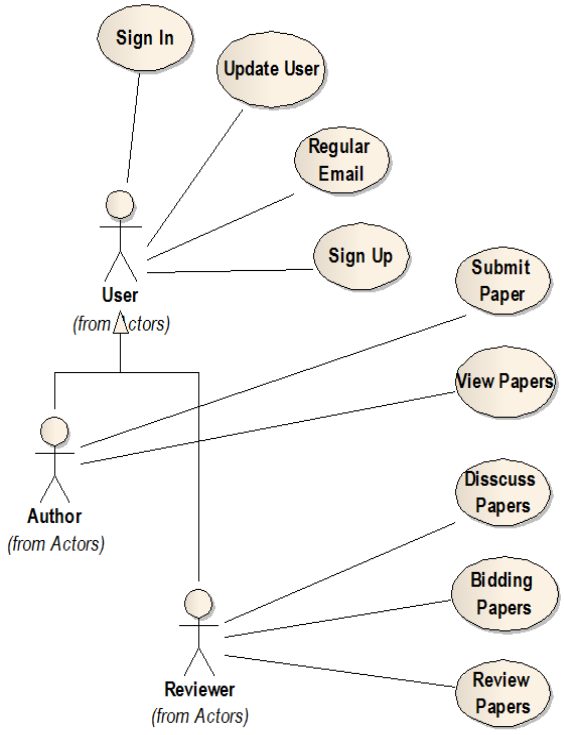


Figure 2: Use Case 2

A. *System Characteristics*

The system we propose is a web-based information management system supporting conference management. Four user groups are defined in the Online Conference Management System:

- System administrator: The system administrator is the person who installs, updates, and restores the system. He/she is also in charge of setting up an account for chairs if their conference request is accepted
- Program chair: The person who makes the overall decisions including assigning papers to reviewers.
- Reviewer: Reviewer is a member of the program committee assigned by the chair to review authors' submissions.
- Author: The person who submits abstracts and full papers.

The Online Conference Management System facilitates the following:

- Covering all the stages of conference management life cycle.
- Allowing the author to submit papers in multiple file formats.
- Automatically assigning papers to a reviewer based on his/her area of expertise and interest.
- Enabling reviewers to make comments and discuss a paper remotely.
- Supporting a group email function.
- Providing phase management support to allow the chair to open and close a conference phase depending on his/her needs.
- Permitting the chair to produce templates for group emails.
- Providing multiple statistics to help the chair in monitoring the review process.
- Granting conference session management to the chair to help him/her prepare the conference agenda.
- Offering a search function to let the chair search for varied information based on varied search criteria.
- Hosting installation on the server. Conference organizers are relieved from problems associated with downloading, installation, and configuration.

B. *System Architecture*

A number of design styles and patterns, such as the Model View Controller (MVC), Presentation–

Abstraction-Control, Philips and REST, are available. [10], [17] and [18]. The proposed system follows the 3-tier architecture. The 3-tier approach encompasses the presentation, the business logic, and the data store layers. Each tier is relatively independent. The application in one tier can request service from a tier below it. To connect different tiers, different connectors are used. Figure 3 illustrates the system's architecture.

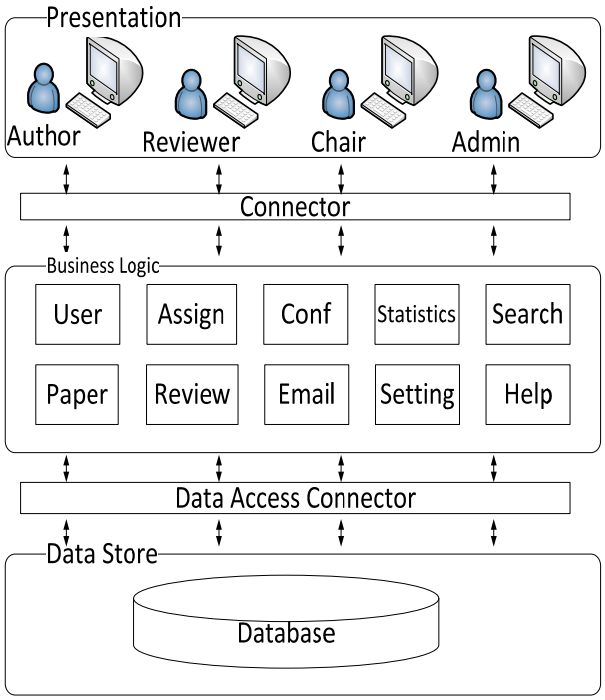


Figure 3: System Architecture

The system's business logic layer consists of 10 components. These components can be further decomposed into several modules. Each module focuses on a specific kind of task. Our goal is to make these components task-independent and easy to reuse. This will enhance the flexibility and maintainability of the entire system. The components are briefly explained below.

User Component: The User Component encompasses two modules; the *Sign in* module to handle the user's sign-in process, and the *Profile* to allow system users to manage their personal information.

Paper Component: This component consists of two modules. The *Edit* module allows authors to edit the abstracts of their papers, and the *Upload* permits authors to upload papers (files).

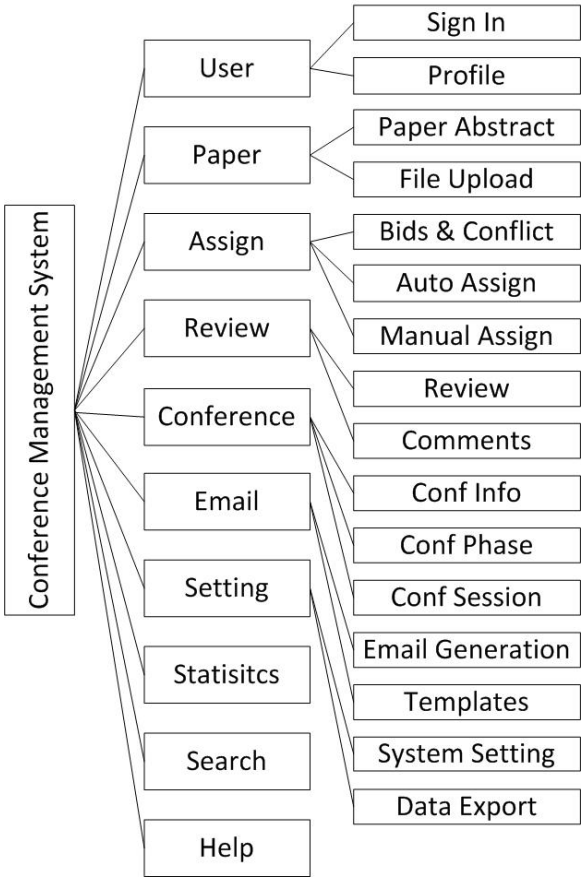


Figure 4: Decomposition of Business Tiers

Assign Component: The Assign component incorporates three modules. The *Bids & Conflicts* module allows reviewers to determine which papers they want to review, and indicate any conflict of interest. This module is optional and can be left out if papers are assigned to reviewers based on other criteria. The *Auto Assignment* module focuses on automatically assigning papers to reviewers based on their expertise and interests. The *Manual Assignment* module allows the chair to override the *Auto Assignment* results and make the final paper assignment decision.

Review Component: This component embraces two modules. The review management is taken care of by the *Rev-Manage* module, and the *Com-Manage* module focuses on comments management, making reviewers' comments available for discussion.

Conference Component: The Conference Component is comprised of three modules. Conference information is handled by the *Information* module. The *Phase* module helps the chair to open/close a conference phase. The *Session* module facilitates conference session management.

Email Component: This component is composed of two modules; the *Template* module is used to help the chair to manage email templates, and the *Composition* module is responsible for composing and sending emails to different users.

Setting Component: The Setting Component includes two modules. The *Change* module helps the chair to change the system's setting. Exporting data is the responsibility of the *Export* module.

Statistics Component: This component provides different statistics to the chair, such as the count of authors and reviewers, and bidding information statistics.

Search Component: The Search Component returns search results to the chair based on search criteria.

Help Component: This component provides detailed help information to assist users in using the system.

C. Database Design

As a core of information exchange and processing, database plays an important role in information systems. It forms the foundation of web services and web-based systems [8]. A very large number of computer applications are database related, and almost every web-based application uses databases to store information. A good database facilitates almost every aspect of an information management system [9].

In our design, the MS SQL Server 2008 is used since it can fully support the .NET platform. Our Online Conference Management System's database is constructed of several database tables, which store various data. Furthermore, a number of stored procedures, which encapsulate the operations on database tables, are used. Table 1 shows the main database tables used in the online conference management system.

D. User Interfaces

User interface plays an important role in any software product. It is through this component that user interaction with the system takes place. In this section, the interface design of the Online Conference Management System is explained. The goal is to make the user interface clear, simple, and friendly. The tool Enterprise Architect, which was constructed by Sparx Systems, is employed to support the user interface design.

Figure 5 presents the starting page of the Online Conference Management System. Different entries for different users are made available. This will facilitate the log-in process. The web page is diveded into five blocks: the header, main menu, navigation bar, main body, and footnote.

TABLE 1
DATABASE TABLES DESCRIPTION

Table Name	Description
tb_user	Stores the basic user information. A column for "role" is used to distinguish different role group of users (system admin, chair, author and reviewer).
tb_paper	Stores the paper information.
tb_review	Stores the review information. Columns "paper" and "reviewer" are used together to indicate the paper assignment. Column "inner comments" represents the comments which will be used in the reviewer's online discussion.
tb_bids	Represents the reviewer's bid for papers. In the "bids value" column, a number of 0-4 indicates conflict, low interest, mid interest, and high interest for a specific paper.
tb_interest	Represents the reviewer's topics of interest.
tb_topic	Stores topic information.
tb_session	Stores conference session information.
tb_email	Stores email information.
tb_emailtemplates	Stores email template information.
tb_configuration	Stores configuration information, which includes basic conference information, conference phase information, submission, review, and email notification setting options.

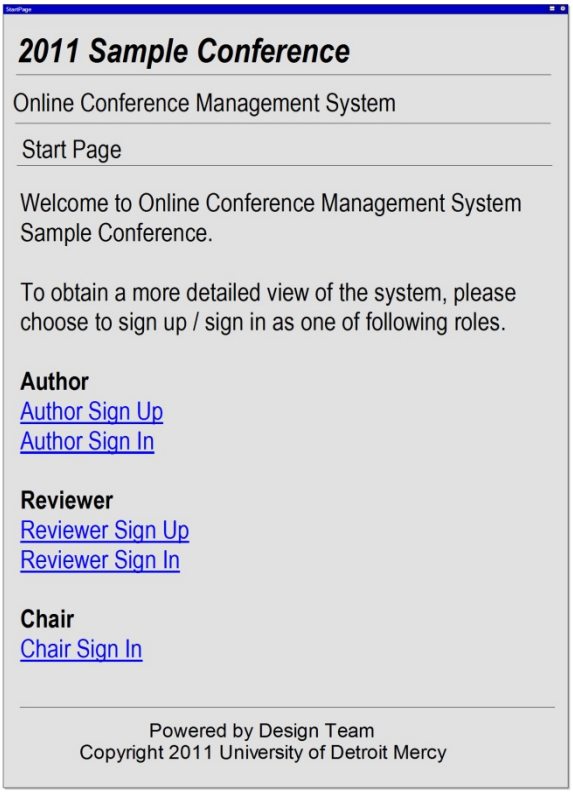


Figure 5: The Starting Page

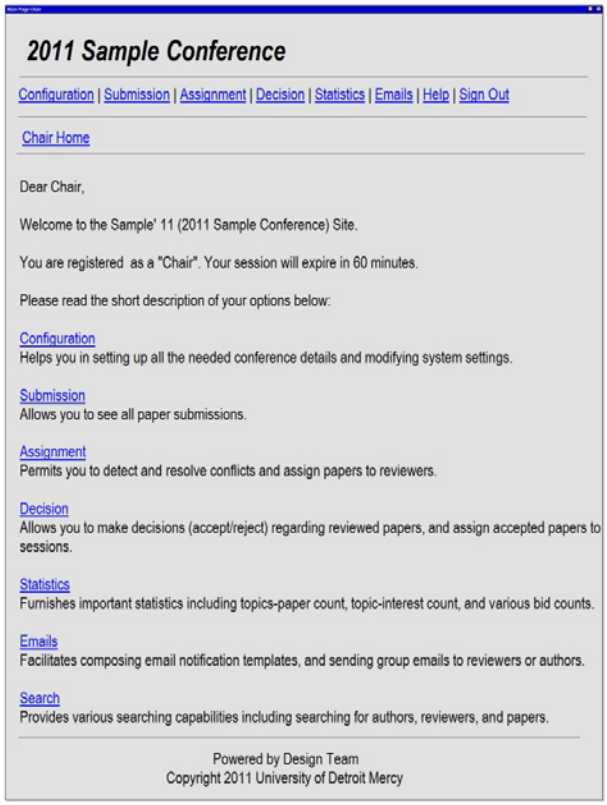


Figure 6: Chair's Main Page

Fiigure 6 illustrates the chair's main page. All the system's pages have the same header and footer, but the contents of the menubar and main body varies. This will help us maintain a unified style of the system's interface. The menubar of the chair's main page contains the following options:

- **Configuration**: Using the confiruation page, the chair can set up conference information and configurae system settings.
- **Submission**: From the submission page, the chair can view all the papers submittd by different authors.
- **Assignment**: Through the assignment page, the chair can check for and resolve conflicts, and assign papers to reviewers.
- **Decision**: This page allows the chair to see all the reviews made by reviewers. The chair can make the final decision on whether to accept or reject a paper. It also allows the chair to assign accepted papers to conference sessions.
- **Statistics**: This page permits the chair to observe a variety of statistical information, such as bids for papers, bids for contribution, conflicts, and paper submission according to the topic of interest.
- **Emails**: Through the email page, the chair can modify email templates, and send emails to a spcific user or a group of users.
- **Search**: At the search page, the chair can search for papers, reviews, and comments using different searching criteria.

IV. Conclusions

With the increase in the number of academic conferences, in the number of papers submitted to such conferences, as well as the complexity of managing such conferences, it is critical to promote an online conference management system that facilitates the task of conference organization using the software engineering process. This paper presents the requirement analysis and design of An Online Conference Management System. Both functional and non-functional requirements are examined. The architecture and data modeling of the system are introduced. Furthermore, a user interface design is proposed and is being implemented. A C# implementation of the system is in progress. For our final version of the system, we plan to improve the system's design to support multi-track conferences and multi-conferences.

References

[1] J. Arlow, I. Neustadt, *UML 2 and the Unified Process,* Upper Saddle River, NJ: Pearson Education, 2005, ch.1 and 4.

[2] M. Franklin, "Rethinking the Conference Reviewing Process," *in Proc. 2004 ACM SIGMOD International Conference on Management of Data*, New York, 2004, pp. 957-957.

[3] M. Huang, Y. Feng, and B. Desai, "CONFSYS2: An Improved Web-Based Multi-Conference Management System," *in Proc. 2nd Canadian Conference on Computer Science and Software Engineering,* Montreal, Canada, 2009, pp. 155-159.

[4] M. Huang, Y. Feng, and B. Desai, "CONFSYS: A Web-based Adademic Conference Management System," *in Proc. Canadian Conference on Computer Science and Software Engineering,* Montreal, Canada, 2008, pp. 141-143.

[5] P. Noimanee, and Y. Limpiyakom, "Towards a RESTful Process of Conference Management System," *in Proc. International Multi Conference of Engineers and Computer Scientists*, Hong Kong, 2009, pp. 991-995.

[6] S. L. Pfleeger and J.M. Atlee, *Software Engineering Theory and Practice,* Upper Saddle River, NJ: Pearson Higher Education, 2010, ch. 4- 6.

[7] R. N. Taylor, N. Medvidvic and E. M. Dashofy, *Software Architecure Fondations, Theory and Practice,* Hoboken NJ: John Wiley & Sons, 2010, ch. 3-4.

[8] G. Post and A. Kagan, "Database Management Systems: Design Considerations and Attribute Facilities," *Journal of Systems and Software*, Vol. 56, No. 2, pp. 183-193, Mar. 2001.

[9] R. Stephons, *Beginning Database Design Soultions*, Indianapolis, IN: John Wiley, 2008, ch. 1-3.

[10] P. Naur, and B. Randell, "Software Engineering: Report of a Conference Sponsored by the NATO Science Committee," *in Proc. NATO Software Engineering Conference,* Garmisch, Germany, 1968, pp.9-65.

[11] J. A. Wang, "Towards Component-Based Software Engineering," *Journal of Computer Science in College,* Vol. 16, No. 1, pp. 177-189, Oct. 2000.

[12] B. Nuseibeh, and S. Easterbrook, "Requirement Engineering: A Roadmap," *in Proc. 22nd International Conference on Software Engineering*, Limerick, 2000, pp. 35-46.

[13] A. Aurum, and C. Wohlin, "The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process," *Information and Software Technology*, Vol. 45, No. 14, pp. 945-954, Nov. 2003.

[14] J. Nicolás, and A. Toval, "On the Generation of Requirements Specification from Software Engineering," *Information and Software Technology,* Vol. 51, No. 9, pp. 1291-1307, Sep. 2009.

[15] J. Bosch, and L. Lundber, "Software Architecture – Engineering Quality Attributes," *The Journal of Systems and Software*, Vol. 66, No. 3, pp. 183-186, Jun. 2003.

[16] E. Folmer, and J. Bosch, "Architecting for Usability: a Survey," *Journal of Systems and Software*, Vol. 70, No. 1-2, pp. 61-78, Feb. 2004.

[17] L. Bass, and B. E. John, "Linking Usability to Software Architecture Patterns Through General Scenarios," *Journal of Systems and Software*, Vol. 66, No. 3, pp. 187-197, Jun. 2003.

[18] R. N. Taylor, and A. Hoek, "Software Design and Architecture: The Once and Future Focus of Software Engineering," *in Proc. 2007 Future of Software Engineering*, Washington, DC, 2009, pp. 226-243.

[19] R. N. Ferrari, and N. H. Madhavji, "Architecting-Problems Rooted in Requirements," *Information and Software Technology*, Vol. 50, No.1-2, pp. 53-66, Jan. 2008.

[20] A. Tang, A. Aleti, J. Burge, and H. Vliet, "What Makes Software Design Effective?" *Design Studies*, Vol. 31, No. 6, pp. 614-640, Nov. 2010.

SESSION

SOFTWARE AND SYSTEM REQUIREMENTS ENGINEERING

Chair(s)

TBA

Nurturing Systems Thinking: An Empirically Based Framework to Improve Systems Development Processes

Arjun Vijayanarayanan and Kelly Neville
Embry-Riddle Aeronautical University
Department of Human Factors and Systems
600 S. Clyde Morris Blvd.
Daytona Beach, FL

Abstract - *In systems development, engineering teams tend to focus on schedule, cost, and technology components, all of which are salient elements of the system. Importance and attention must also be given to nonsalient system elements, the neglect of which may contribute to the failure of large systems engineering projects. In this effort, a framework is developed to improve quality in systems engineering processes. The framework can be used to help developers expand their attention and efforts beyond the salient aspects of the system development process. Qualitative research methods were used to identify the technical, teamwork, management, and organizational factors that can affect the quality of a system development project. More specifically, interview data were assessed in a bottom-up manner to identify emergent patterns and in a top-down manner to evaluate the relevance of Human Factors Analysis and Classification System (HFACS) factors, a framework used to improve safety in the aviation industry.*

Keywords: Systems development, systems thinking, qualitative research, quality

1 Background

Late in his career, accomplished computer scientist and mathematician Joseph Goguen turned his attentions to the relations between social and technical factors in systems development. Goguen [8] noted that "large projects have an embarrassingly high failure rate" (p. 165), and his observation is supported by statistics produced by the Standish "CHAOS Report", which provides information on software project failures and the factors that lead to failure. The project failure rates reported during 1994-2009 can be seen in Table 1.

The statistics are especially alarming in light of the costs of large systems and software engineering projects. For example, the US Internal Revenue Service (IRS) spent $4 billion on a system that, as described by an IRS official, does "not work in the real world," [16] and the US Federal Bureau of Investigation spent $170 million on a system overhaul that ultimately they abandoned [5]. Other well-known system development failures involved upgrades to the London emergency dispatch system [7] and the US air traffic control system [4].

Table 1

The Standish Group's "CHAOS Report" Showing Information Technology (IT) Project Failures and Successes from 1994-2009

Year	Failed Projects	Challenged Projects	Successful Projects
1994	31%	53%	16%
1996	40%	33%	27%
1998	28%	46%	26%
2000	23%	49%	28%
2004	18%	53%	29%
2006	19%	46%	35%
2009	24%	44%	32%

Note. Adapted from [6].

Goguen [9] implicates as a root cause of the very high failure rate the neglect of human and social aspects of systems during engineering projects. Additional evidence of this tendency to neglect human and social aspects can be found in the effort to add a human view to the Department of Defense Architectural Framework (DoDAF) [1], [3], [10]. Similarly, there is no accepted engineering practice, procedure, or modeling formalism for designing user interfaces that support the work and work practices of the users. Robert Hoffman, a human factors methodologist, notes that popular software and system engineering approaches (e.g., waterfall, spiral, and agile development frameworks) are noticeably lacking in

guidance for integrating technology designs with their users and use environments (for example, see [11]). In the following sections we will discuss two approaches: Goguen's ethnomethodological approach and Systems thinking approach, which illustrate the importance of the social aspects of systems.

1.1 Goguen's Ethnomethodological Approach

Goguen used ethnomethodology as a tool to study the social aspects of a system. Ethnomethodology is a method for understanding the ways in which culture shapes and imbues the work practices, goals, and priorities of people in an organization by analyzing their accounts of their day-to-day experiences, work artifacts, and their communications. It is a descriptive method and does not engage in the explanation or evaluation of the particular culture undertaken as a topic of study. Goguen [9] states, "Ethnomethodology tries to reconcile radical empiricism with the situatedness of social data, by looking closely at how competent members of a group actually organize their interactions" (p. 10).

Goguen uses ethnomethodology to explain the *principle of accountability* and *the principle of orderliness.* According to the principle of accountability, the members in a group are held accountable for their actions depending upon where their group is placed in the society or, in our context, in the organization. Thus, the behavior and interaction of members of a work group are constrained by the nature of accountability imposed by the group. According to the principle of orderliness, social interaction and behavior are orderly and can be understood with respect to contextual and cultural constraints. Thus, a group's interaction can only be fully understood in the context of that particular group, which is the essence of 'qualities of situatedness'. Thus to understand the nature of a system, the work group should be analyzed. The failure to take into account this human side of the system leads to system designs that are poorly matched to an organization's work culture and practices. Thus, ethnomethodological approaches play a vital role in the development and achievement of *systems thinking*, an important systems engineering perspective that will be explained in the next section.

1.2 Systems Thinking Approach

According to International Council on Systems Engineering (INCOSE) [13], a system is "a construct or collection of different elements that together produce results not obtainable by the elements alone. The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results" ("Definition of a System", para. 1). Note that people are included in this definition as a core system element. Thus although people and related social factors tend to be neglected by systems developers, the systems engineering profession identifies them as key factors to be addressed. The systems engineering profession emphasizes the importance of *systems thinking* - the consideration of a system in a holistic way, in terms of all its components and their relationships. To help developers engineer systems in ways consistent with the goal of systems thinking, a framework called the *Systems Thinking Framework for Systems Development (STFSD)* is being developed.

The idea of the systems thinking framework came from the Human Factors Analysis and Classification System (HFACS) [22]. HFACS is a safety assessment framework widely used in aviation accident investigation because it helps analysts consider the latent failures within a web of causal influences. A similar framework in systems development could help developers monitor whether they are engineering the system in a balanced way; that is, whether they are comprehensively addressing the factors that affect the quality of a system development team's work and whether they are using processes, methods, tools, etc. that support a balanced and holistic approach. The framework should help developers attend to development activities that address human aspects of systems as well as they attend to the technical aspects. The framework's development is rooted in the premise that helping developers attend comprehensively to development activities, including activities that address the human (or social) aspects of systems, will lead to products in which humans are well integrated.

2 Method

2.1 Data Collection

In a recent issue of the journal *Systems Engineering,* [21] lament the lack of empirical research in the field of systems engineering. Methods they advocate for studying systems engineering include semi-structured and unstructured interviews, qualitative research methods that were used in the present research because they produce rich, relatively unbiased (i.e., less influenced by the researcher) data. This empirical research was conducted by analyzing the transcripts of six semi-structured interviews with experienced systems engineers.

The semi-structured interviews analyzed for this research were collected by [12]. Hoffman et al. analyzed these data to identify challenges faced by systems

engineering teams; in the proposed work, the data were re-analyzed to identify systems development practices, activities, and other factors involved in the conduct of a development effort. The identified factors serve as an initial solution for the proposed framework.

Hoffman et al. conducted semi-structured interviews with six systems development practitioners. Four were trained as software engineers and the other two were trained as systems engineers. Five of the engineers had 20 years or more of experience and one had 15 years of experience. Each engineer was asked to recall a challenging engineering project and to talk through it, describing challenges faced and strategies used to overcome them. This approach is based on the Critical Decision Method of knowledge elicitation [15]. Interviews were approximately 2 hrs in duration. Interview data were audio recorded; audio recordings were subsequently transcribed.

2.2 Data Analysis

To analyze the interviews, the transcripts were broken down into data elements, where each element represented a low-level unit of information—a specific idea, claim, or description. Data analysis methods focused on identifying development teams' goals, activities, concerns, and other influences on development work. Data analysis was conducted using bottom-up and top-down coding approaches. The top-down approach allowed us to evaluate the relevance of organizational factors used in HFACS; the bottom-up approach allowed us to identify factors that seem to be suggested by the data.

To conduct the bottom-up coding, data were first reviewed broadly to identify themes that emerged. An initial set of data-driven codes (categories of information found in the data itself) was developed based on these early emergent themes. Part of the data was coded using this initial set of codes; as they were used, the codes were adapted to achieve a better fit with the patterns in the data. This adapted set of codes was then used to code all data. The codes used with all the data were relatively stable but nevertheless continued to evolve through the entire process. When a data element could not be coded using the existing bottom-up codes, a new code was identified and added to the set of bottom-up codes. Data driven codes fell into categories that included project management, work conditions, and teamwork strategies.

In the case of the top-down coding, HFACS categories that represented organizational influences were used as codes. HFACS-based codes were dropped if they were not found to be relevant to any interview data. These HFACS categories used to code included factors such as organizational process, organizational climate, and resource management.

Multiple coders (three in number; Coders A, B, and C) were trained to code the data. Coder A performed both the data-driven coding and coding using the HFACS categories, Coder B performed the data-driven coding and Coder C performed the HFACS coding. To assess the reliability of coding, the percent of correspondence between each coding pair was determined. Differences in codes were reconciled by Coder A. Thus, Coder A used each second set of codes to help him consider alternative interpretations and perspectives.

3 Results and Discussion

Percentages of data elements assigned the same code by both coders in a pair were 74.71% for data-driven codes and 81.81% for HFACS codes. The final sets of codes that emerged from the coding analyses became the factors that comprise the first version of the systems thinking framework.

3.1 Organization of the Framework

The Systems Thinking framework for Systems Development consists of four tiers, namely *organizational influences*, *project management*, *team processes* and *technical activities*. Each tier has a set of factors associated with it. The factor *external pressures* envelopes all the tiers. The hierarchical representation of the systems thinking framework shown in Figure 1 specifies four major sources of influence on the technical activities tier. These influences are changes and associated pressures in the three tiers above: external pressures, organizational influences, project management, and team processes. These sources of influence are organized in the framework according to the extent to which they interface directly with and directly impact the technical activities. The structure represents the way in which different layers of an organization can influence the way work is conducted, a concept borrowed from the HFACS framework.

3.2 The Emergent Tier

When people think of the structure of a system development organization, it is common to think of it as an organization led by executives, where project managers help oversee or coordinate the technical activities. Support for this conceptualization was found in the interviews, however, the data suggested a fourth, unanticipated tier. In particular, the data contained a great deal of information regarding teamwork. Teamwork taxonomies by [17] and [20] were identified and used to perform a "finer-grained" analysis on teamwork. From this teamwork analysis, the following codes, as seen in Table 2, emerged as important.

Table 2

List of Added Codes

Team Coordination and Communication
Mission Analysis, Formulation, and Planning
Goal Specification
Team Monitoring and Back-Up Behavior
Coordination
Conflict Management
Initiative
Information Exchange
Build Common Ground and Awareness

Figure 1 presents the high-level factors derived from the coding analysis. All codes used in the analysis are included as factors in the framework, as long as they mapped to data in at least one interview. Codes that mapped to data in only one or two of the interviews were kept in the framework; weak evidence for them in this particular study does not disprove their relevance to systems engineering. Future research can shed more light on their potential roles and, in the meantime, their inclusion in the framework may enrich the framework's support for systems thinking. More firmly established factors are factors that were supported by mappings to data in three or more of the interviews.

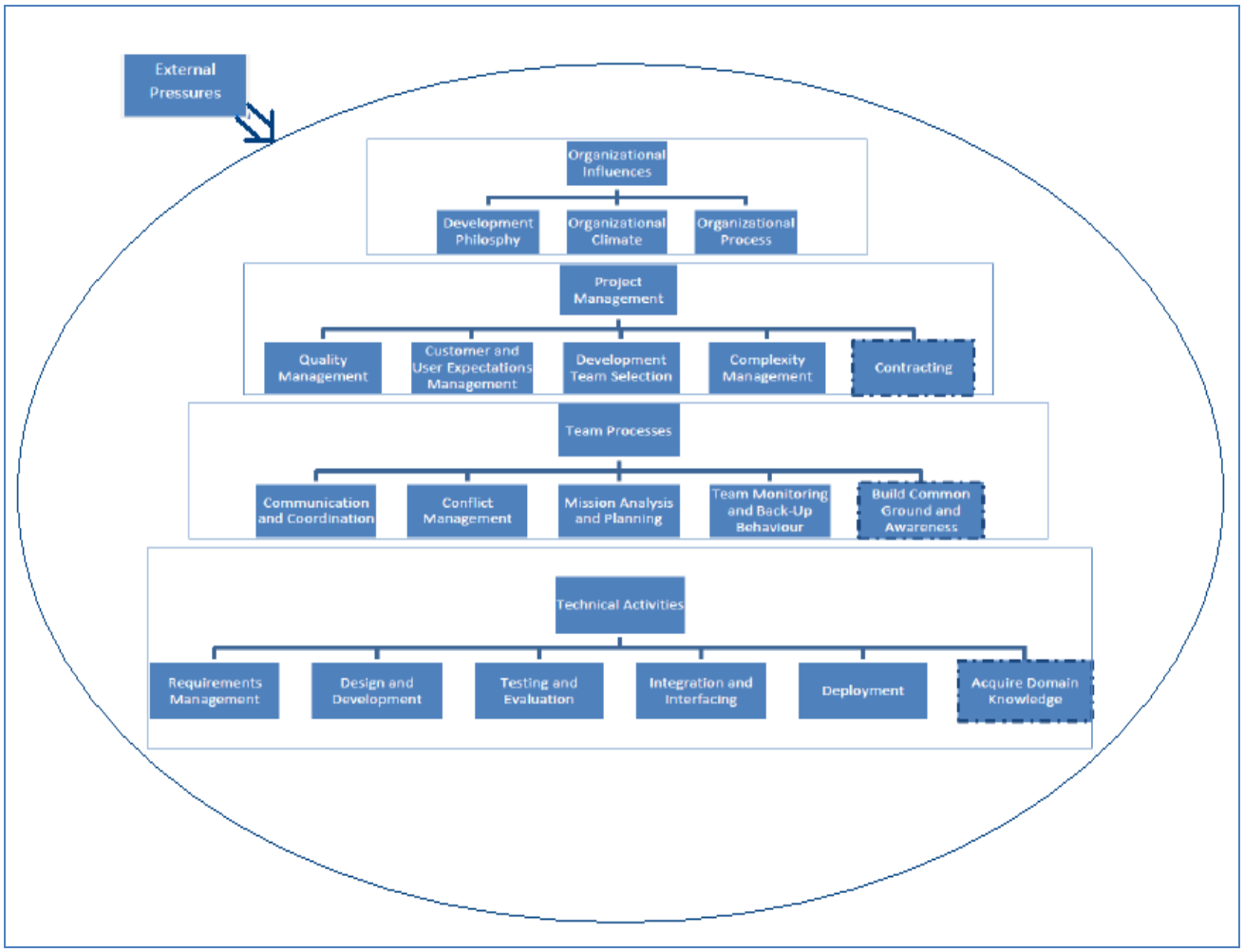


Figure 1. The Systems Thinking Framework for Systems Development (STFSD). *Note.* Dashed lines around nodes indicate that a factor was found in only one or two of the six interviews. All the other factors were found in at least three of the interviews.

3.3 Encouraging Attention to Human Aspects of Systems

A main goal behind developing the systems thinking framework was to help developers attend to the large number of factors that can influence the outcome of their work and, especially, the factors that can influence how well a development team attends to social aspects of systems and to the integration of social and technical aspects. To this end, framework factors that can contribute directly to the integration of social and technical aspects have been identified post hoc. These factors consist of 'Acquire Domain Knowledge' on the 'Technical Activities' level, all teamwork strategies identified under 'Team Processes', and the factor 'Complexity Management' under the 'Project Management' node. Justification for selecting these particular factors is as follows:

Acquire Domain Knowledge – Acquiring domain knowledge is a type of technical work that involves learning about the environment and activities in which a system will be used. Acquiring Domain Knowledge is essential to both understanding the human aspects of a system and integrating them with the technical aspects. This component of systems development is where Goguen's recommendation for using ethnomethodology can be brought to bear.

Teamwork strategies – In the interviews, teamwork strategies were typically cited as supporting teamwork among development team members as they worked on a system's technical components; the same strategies could facilitate teamwork between members working on technical and human components of a system. Because people working on technical and human system aspects often are trained in different disciplines guided by differing

systems development philosophies, the teamwork strategies may be especially valuable—even vital—to the success of their working relationships.

Complexity Management - Complexity management encompasses strategies that are used to either reduce or cope with the complexity of a system being developed or of the systems development process. When the human aspects of a system are considered, their consideration tends to introduce a great deal of complexity into the development process and those aspects furthermore represent complexity that is often ignored within the system being developed. Consequently, complexity management activities may be critical to the success of a team that addresses a system's human elements.

In fact, the interview data did not address human aspects of systems or human-technology integration, with very few exceptions. Similarly, the interviewees tended not to relate the three factors listed above to human-technology integration. For example, a domain analysis of the environment and activities in which a system would be used was described in only one interview. One other interviewee used the term 'domain analysis' but it was in the context of gaining an understanding of the engineering domain and engineering precedents. The teamwork factors, when described by the interviewees, tended to relate to conflicts, tensions, team monitoring, information exchange, and shared understanding among team members focused on technical aspects of a system. Complexity management typically involved technology decomposition and technology requirements management.

Support at the organizational and program management levels of the systems thinking framework would likely be required for the three factors above to be used in support of human-focused development activities and human-technology integration. Future research could evaluate this hypothesis and future versions of the systems thinking framework may further develop the roles and influence of these levels. More human-related activities may also be added to the framework as additional research is conducted and the relationships between specific activities and the probability of successful outcome are better understood. Generally speaking, the integration of human and technology elements tends to occur well when development work includes opportunities for human and technology elements to shape each other (for example, see [2], [18]).

4 Conclusions

This research involved the development of a framework for drawing attention to both straightforward and the underlying, latent and more abstract factors that can affect a system development project's quality and chances for success. The framework makes explicit the influences that contribute to tunnel vision during development practices and which thereby put a project at risk. Use of the resulting framework should help development teams to better understand causal factors that can lead to development failure and look out for them in the future. Systems engineering processes and projects should consequently become more effective and more likely to result in a system in which all elements integrate well. Based on this research, we suggest a certain set of factors that systems development teams should monitor; other factors may be added based on future work. Future work should additionally investigate risks and strategies associated with the framework factors so that more specific and useful guidance can be built into the framework.

5 References

[1] Baker, K., Stewart, A., Pogue, C., & Ramotar, R. (2008). *Human views: Extensions to the Department of Defense Architecture Framework* (DRDC CR-2008-001). Defense Techical Information Center, Ft. Belvoir, VA. Retrieved from http://www.dtic.mil.

[2] Benbya, H. & McKelvey, B. (2006). Using coevolutionary and complexity theories to improve IS alignment: A multi-level approach. *Journal of Information Technology, 21*, 284-298.

[3] Bruseberg, A. (2008). Human views for MODAF as a bridge between human factors integration and systems engineering. *Journal of Cognitive Engineering and Decision Making, 2*, 220-248.

[4] Carr, D. F., & Cone, E. (2002, April 8). Can FAA Salvage Its IT Disaster? *Baseline Magazine*. Retrieved from http://www.baselinemag.com/article2/0,1540,656862,00.asp.

[5] Eggen, D. (2005, June 6). *FBI Pushed Ahead with Troubled Software*. Retrieved from *Washingtonpost. com*.

[6] Eveleens, J. L., & Verhoef, C. (2010). The rise and fall of the Chaos report figures. *IEEE Software, 6*, 30-36.

[7] Finkelstein, A., & Dowell, J. (1996). A Comedy of Errors: The London Ambulance Service Case Study. *IEEE Proc. 8th Int'l Workshop Software Specification and Design,* 2–4.

[8] Goguen, J. A. (1994). Requirements engineering as the reconciliation of social and technical issues. In M. Jirotka and J. A. Goguen (Eds.), *Requirements Engineering: Social and Technical Issues* (pp. 165-199). San Diego, CA: Academic Press Professional.

[9] Goguen, J. A. (1997). Towards a social, ethical theory of information. In G. Bowker, L. Gasser, L. Star, & W. Turner (Eds.), *Social Science Research: Vol. 115. Technical Systems and Cooperative Work: Beyond the Great Divide* (pp. 27-56). Mahwah, NJ: Erlbaum.

[10] Handley, H.A.H. & Smillie, R.J. (2008). Architecture framework human view: The NATO approach. *Systems Engineering, 11*, 156-164.

[11] Hoffman, R. R., & Elm, W. C. (2006) HCC implications for the procurement process. *IEEE: Intelligent Systems*, *January/February*, 74-81.

[12] Hoffman, R.R., Neville, K., & Fowlkes, J. (2009). Using cognitive task analysis to explore issues in the procurement of intelligent decision support systems. *Cognition, Technology, & Work, 11*, 57-70.

[13] INCOSE (2006). *A consensus of the INCOSE Fellows*. Retrieved from http://www.incose.org/practice.

[14] Johnson, R.B. (1997). Examining the validity structure of qualitative research. *Education*, *118*, 282-292.

[15] Klein, G., Calderwood, R., & MacGregor, D. (1989). Critical decision method for eliciting knowledge. *IEEE Transactions on Systems, Man, and Cybernetics, 19(1),* 462-472.

[16] Marketplace (1997, January). *Marketplace for January 31st, 1997*. National Public Radio.

[17] Marks, M. A., Mathieu, J.E., & Zaccaro, S. J. (2001). A temporally based framework and taxonomy of team processes. *Academy of Management Review*, *26(3),* 356-376.

[18] Norman, D. O. (July 2004). *Engineering a complex system: A study of the AOC* (MITRE Technical Report No. 04-0527). MITRE. Retrieved from http://www.mitre.org/work/tech_papers/tech_papers_04/norman_aoc/norman_aoc.pdf.

[19] Paletz, B. F., Bearman, C., Orasanu, J., & Holbrook, J. (2009). Socializing the Human Factors Analysis and Classification System: Incorporating Social Psychological phenomena into a Human Factors Error Classification System. *Human Factors*, *51*, 435-445.

[20] Smith-Jentsch, K.A, Zeisig, R.L, Acton, B., & McPherson, J.A. (1998). Team dimensional training: A strategy for guided team self-correction. In J.A. Cannon-Bowers & E. Salas (Eds.), *Making Decisions under Stress: Implications for Individual and Team Training* (pp. 271-297). Washington, DC: APA Press.

[21] Valerdi, R., & Davidz, H. L. (2009). Empirical research in systems engineering: Challenges and opportunities of a new frontier. *Systems Engineering, 12*, 169-181.

[22] Wiegmann, D. A., & Shappell, S. A. (2003). *A human error approach to aviation accident analysis: The Human Factors Analysis and Classification System*. Burlington, VT: Ashgate Publishing Company.

A Practice of UML for Web Development

Kuang-Nan Chang[1] **and Peter Henderson**[2]
[1]Department of Computer Science, Eastern Kentucky University, USA
[2]School of Electronics and Computer Science, University of Southampton, UK

Abstract – *This paper proposes a method that uses UML diagrams as a software development process for analysis and design of web applications. The method uses the most needed diagrams only to analyze system requirements and perform software design in a very short term. The method starts with a use case diagram to define functional requirements of the system. Workflow diagrams follow to specify flows of activities of those identified requirements in the use case diagram. Class, sequence, and activity diagrams are then used to determine necessary classes, functions, and algorithms that need to be implemented. The method have been applied to two projects and proved its usefulness in web development.*

Keywords: web application, web development, UML.

1 Introduction

Today almost all commercial companies, educational institutions, governmental organizations, and the other businesses put their information online to extend their markets and/or broadcast their information. Web engineering for developing such web applications and its extended web-based mobile applications, has become an important subject.

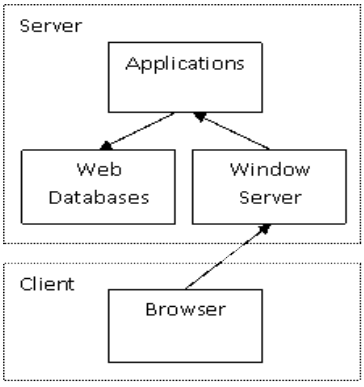


Figure 1. Basic System Architecture for Web Applications

Figure 1 shows a possible system architecture of web applications. Users at the Client side use browsers to communicate with the Window Server at the Server side. Requests are sent to the Window Server, where applications perform computing calculation and send the results back to the users. Web Databases are set for data storage and access. Other components can be added to the architecture [4, 5, 6]. Figure 1 just shows the very basic architecture that a web application can have.

UML, a design and communication tool for analyzing and designing object-oriented systems has been applied to software construction in many domains and projects. Web development is one of these new domains where people try to take its advantages. Although it is not necessary to use UML or OO in web development, in many textbooks and research papers [2, 9], authors emphasize the importance of using OO for web development. Almost all of them agree that agile process models are more appropriate for web development, because they have shorter development time and accommodate to changes. With such process models, techniques such as UML can be used to help specify system requirements and design quickly.

It causes our attentions that a link between basic system architecture such as shown in Figure 1 and agile process models needs to be expressed specifically. This link should meld the implementation of the architecture with a short term process model. With UML, the link is constructed with its different diagrams for specify system requirements and software design issues.

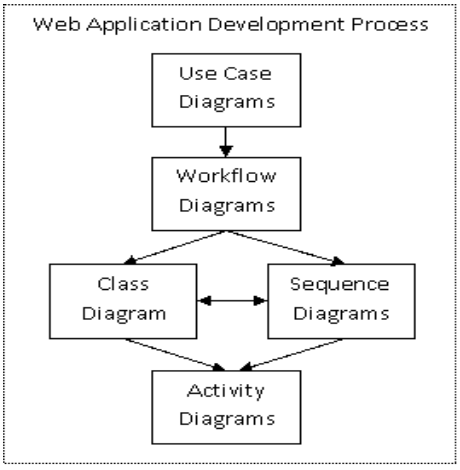


Figure 2. Web Application Development Process

This paper illustrates a practice of UML for web development. It proposes a method that uses UML diagrams as a software development process for analysis and design of web applications. Figure 2 shows the proposed development process for web development. The method uses the most

needed diagrams only to analyze system requirements and perform software design in a very short term. The method starts with a use case diagram to define functional requirements of the system. Workflow diagrams follow to specify flows of activities of those identified requirements in the use case diagram. Class, sequence, and activity diagrams are then used to determine necessary classes, functions, and algorithms that need to be implemented. The method have been applied to two projects and proved its usefulness in web development.

The rest of the paper is organized as follows. Section 2 introduces the proposed method for web development. Section 3 discusses the uses of UML diagrams and other techniques in the method in details, including consistency checks of the UML diagrams. Section 4 shows two projects that use the method and complete two web applications, and section 5 concludes the paper.

2 Web Application Development Process

This section introduces the proposed method, which melds the simplified UML process model with the basic web system architecture, described in the previous section, together for web development. Figure 3 shows the idea of the method. The web development process at the right hand side of the figure supports the development of all the system components in the web architecture at the left hand side.

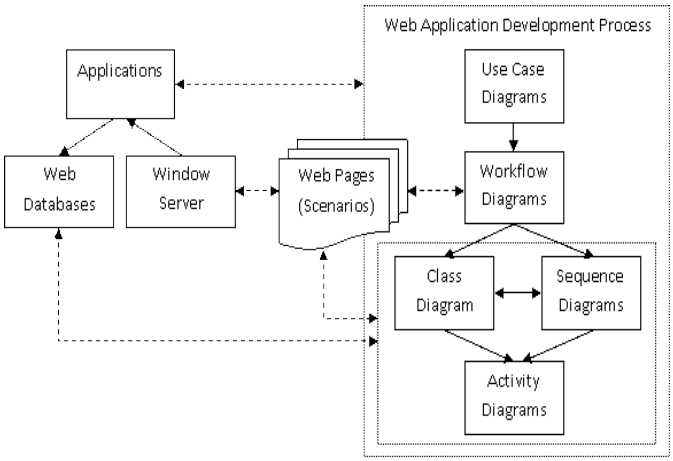


Figure 3. Web Application Construction

In the process, first a use case diagram is used to specify the needs and requirements of a web application with various use cases. Each use case represents a complete business activity. An activity can be defined as a scenario, which is a synthetic description of an event or series of actions and events. To catch these scenarios, we use workflow diagrams to illustrate the events, actions, and their branches of the scenarios. For example, considering a simple logon scenario, both successful logon and unsuccessful logon cases should be expressed in a workflow diagram. Although Fowler [3] indicates that using activity diagrams to specify use cases may not be easy to follow them, there are two major advantages of using workflow diagrams to specify the scenarios of use cases: abstraction and soundness checking. When using activity diagrams to describe the activities, they can focus on their flows of events without specifying classes involved in the events. This feature allows us to specify the behaviors of a system from a more abstract viewpoint, especially for web applications in which events and activities are often specified before classes/objects identification. The other advantage is able to check the soundness of specified activities whether they are sufficient enough to achieve the tasks specified in the use cases. Soundness checking such as dead tasks, and the tasks that causes deadlocks and livelocks, should be identified and prevented in the diagrams [8].

After creating workflow diagrams for the system, the next step can be divided to two branches: one is to design web pages and use them to do scenarios analysis, and the other is to create a class diagram and a set of sequence diagrams. Web page design plays two important roles in the process. First the web pages are used to construct the interfaces of the window server in the system architecture. With the web pages, customers are able to have an early vision of the final product as well as how users will interact with the system. Second the web pages can be used to verify the actions/events in the workflow diagrams. They can help in scenario analysis and check the soundness of the diagrams with a series of graphic user interface displays. We have found that web page design is very important in web development, and it is the first bridge that connects web system architecture and web development process.

The bottom part of the development process concerns the implement issue of the system, which include a class diagram, different sequence diagrams, and activity diagrams. They are related to the components (classes) and their functions that support the services provided in web pages. Web databases provide storage media in which data accesses are implemented in the functions of components.

With workflow diagrams, we can specify classes that are needed and how they interact with each other to accomplish the tasks in the scenarios. A class diagram and a set of sequence diagrams are created next, to show the static system architecture and the dynamic behaviors of the system respectively. These diagrams are constructed simultaneously for saving some development time and for referring to each other.

With identified classes in the class diagram and messages (functions) in the sequence diagrams, the next step is to create various activity diagrams to put details of the messages and classes. Objects in the sequence diagrams are used to create swimlanes, and the sequences of messages are expressed with flows of actions/events, which will be mapped to relevant coding statements later. Some activities in the activity diagrams need to access web databases. Together with the

class diagram and the sequence diagrams, they can help to design the schema and queries of the web database system.

The five UML diagrams used in the proposed web application development process support the development of the applications that are needed in the web system. Web page design and web database system design can be done with the constructions of these diagrams.

3 Consistency Checking of UML Diagrams

Consistency checking serves two purposes of software development with UML: it keeps the consistency between the diagrams of UML, and it improves the correct uses of symbols and the completeness of the diagrams [3, 8]. Only when these diagrams are consistent and complete, they are able to express effectively different aspects of a software system [3, 9].

3.1 From Use Cases to Work Flow Diagrams

G1: Make sure that each use case has a corresponding workflow diagram to express the scenarios covered in the use case.

A use case is just a definition of a system service, or a business activity, requested by customers. A specification of such a defined service is descried with a workflow diagram. Guideline G1 is to make sure that each complete service has a definition part (use case) and a specification part (workflow diagram). For example, a possible use case, Make Payment, is a requested service for an online purchase. A workflow diagram should be used to specify the flows of activities necessary to complete the service. These two diagrams together complete the description of a functional requirement of a system.

G2: Make sure that each workflow diagram completely covers all of the scenarios that a use case intends to have.

This guideline suggests that every scenario and its branches of a use case should be covered in a workflow diagram. With the previous Make Payment use case, it may allow customers to make a payment with credit cards, store cards, or checks. All of the three methods may or may not pass the validation of the cards or checks. A workflow should cover all the three scenarios and their two possible branches. If the system allows customers to use the combination of the three payment methods, the workflow diagram also needs to specify such combination of scenarios.

3.2 From Workflow Diagrams to Class Diagram and Sequence Diagrams

G3: Make sure that each workflow diagram has at least one sequence diagram to express its activities with sequences of messages (function calls).

In our method, a sequence diagram is used to express a flow of activities in a workflow diagram with a sequence of messages (functions) that are needed to complete a scenario. For example, when using a credit card to make the payment in the example, a sequence diagram may contain the following sequence of messages: readCard(), validateCard(), and displayConfirmation() (if the card passed through the validation) or diaplayError() (if the card didn't pass through the validation). Guideline G3 requests that each work flow diagram should have at least one sequence diagram to specify necessary functions to cover a scenario in the diagram.

G4: Make sure that all the classes have been used in one or more sequence diagrams. If any class is not used, make sure that it is for reusability or other purpose.

G5: Make sure that all the classes in the class diagram have correct relationships to interact with each other in the sequence diagram.

Guidelines G4 and G5 are from our previous work [1]. The purpose of these two guidelines is to make sure that each class is necessary and used in the system, and their relationships are consistently expressed in both the class diagram and the sequence diagrams.

3.3 From Class Diagrams and Sequence Diagrams to Activity Diagrams

G6: Make sure that each sequence diagram has a corresponding activity diagram to illustrate each message passing (function call) with flows of actions/events.

G6 indicates that a sequence diagram is used to construct a corresponding activity diagram. Each message in the sequence diagram is expressed in detail with flows of actions/events that are needed to fulfill the requirements of the message. Take the readCard() message in G3 as example. Actions that may take to complete this function include *displaying a form*, *asking customers to enter card information*, and *reading and saving the card information temporary in the system*. If the customers missed some fields of the form, another set of actions should be performed to ask the customers to modify the missing fields. The flow of these actions is like an algorithm design for the function readCard(). They will be implemented in the later coding phase.

G7: Make sure that all the classes have been used in one or more activity diagrams. If any class is not used, make sure that it is for reusability or other purpose.

This guideline is a reflection of G5. It makes sure that all of the classes in the class diagram have their uses in the system.

4 Project Examples

There were two web applications completed by using the proposed method in 2009. Although they were term projects requested in a Web Engineering course, they were originally requested by two offices at Eastern Kentucky University: Kentucky Educational Collaborative for State Agency Children (KECSAC) and the department of Computer Science. Students were divided into two groups to do the projects. They followed the proposed web development process and completed the projects, in good quality, within two months.

4.1 KECSAC Project

This project was to develop a web application for the KECSAC office of EKU to replace and enhance functionality of its original system. The main task of the system was to keep, access, and search various data of Kentucky State schools in a database. The original system was implemented in Microsoft Access. It was installed in a computer, which allowed only one user to access the system at a time. Data were duplicated and kept in the database for different years. The system was not reliable because of many minor errors. It also was not user friendly because users needed to memorize different complex operations to execute different functions.

The office requested a new system that satisfies multi-users, remote access, and efficiency and effectiveness requirements. A team of four students were assigned to do the project. They spent four weeks in learning the proposed method and applying it to the project. They spent another four weeks to implement the system and relevant documents. Currently the system has been used and tested by the office. So far they haven't found any fatal errors and are very satisfied with the new system. The following figures are some screenshots of the system.

With the new system, users from the office can now use the system with any browsers from different sites. Multiple users can login to the system and access the data simultaneously. Figure 4 shows a login page to verify valid users of the system. Figures 5 and 6 show how various data are displayed in a more organized way. Data can also be modified with the same pages. Figure 7 shows the print function of the system, which allow users to generate different report for printing.

Figure 4. KECSAC Login Page

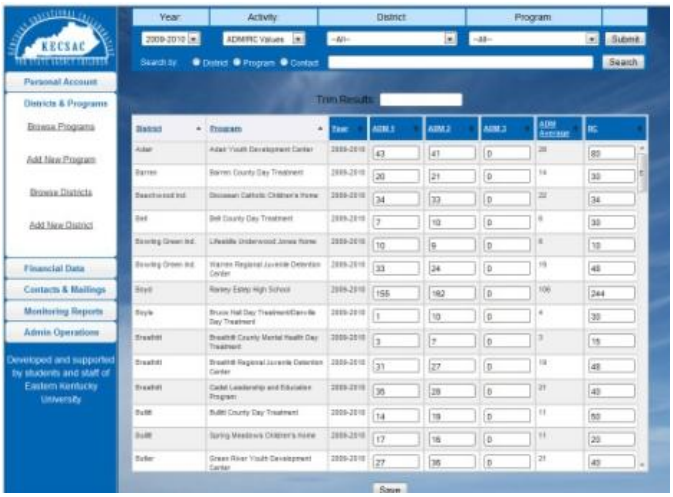


Figure 5. School Activities Display Page

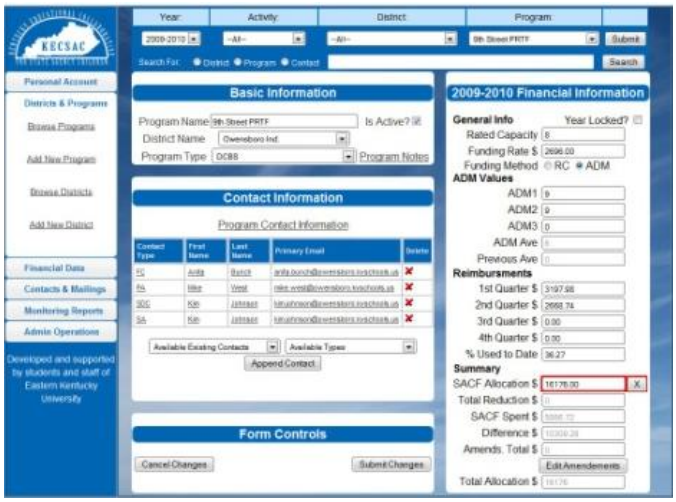


Figure 6. Contact Information Editing Page

Figure 7. Various Reports Generation Page

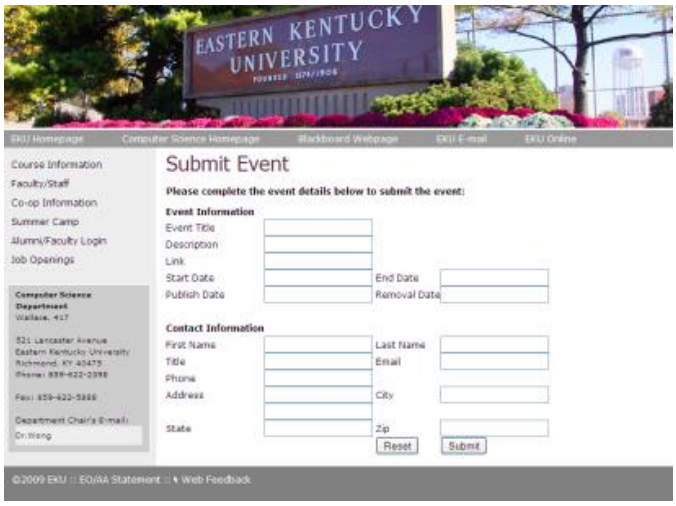


Figure 8. Announcement Editing Web Page

4.2 Computer Science Department Wet Site Project

The main task of this project was to design and implement new web pages for the department of Computer Science of EKU. The department was asked to update its web site so that all the departments and offices would have a uniform look. Other than displaying static departmental information, the department wished to set up an on-line survey for its alumni, create accounts for its alumni for job hunting and resume submission, and create accounts for its faculty members to maintain their own displaying data.

A team of five students were assigned to this project. They spent two months to create a good quality software system for the project. Although the department wasn't able to use the system at the end due to the security policy and server criteria of the university, many faculty members though that the students' system had better performance and quality than the one later used by the department. The following screenshots show some web pages of the system.

Figure 8 shows the web page which allows the department to edit and add new announcements to the web site. Figure 9 shows the page for editing job opportunities. The system also helps companies to find good candidates from the graduates who will be fit for their new positions. Figure 10 shows the page for searching such candidates. Finally, Figure 11 shows the page which allows the graduate to put their resume at the web site.

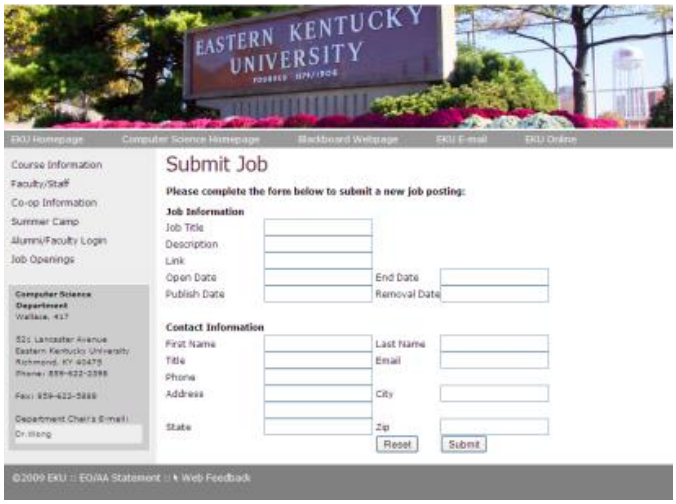


Figure 9. Job Editing Web Page

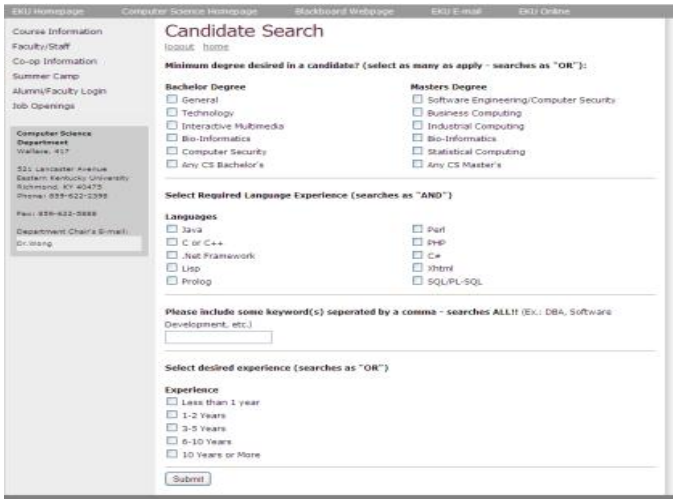


Figure 10. Job Matching Web Page

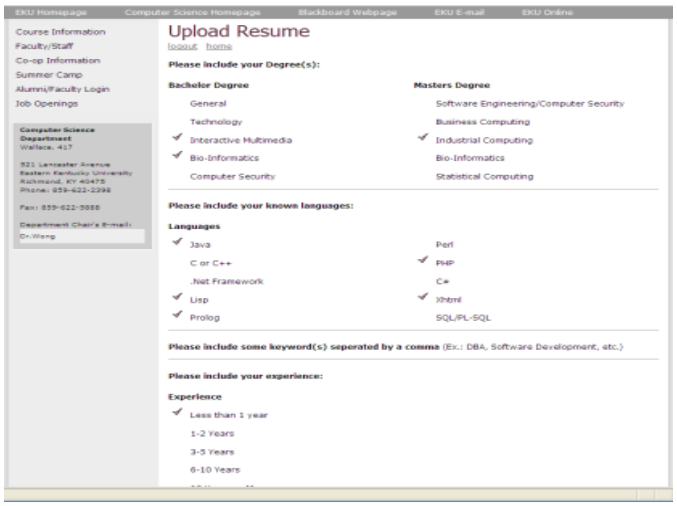


Figure 11. Resume Submission Web Page

4.3 Discussion

Both the KECSAC and CS Department projects have been implemented with the proposed method. They were team projects of a Web Engineering course. At the beginning of the semester, the instructor didn't expect too much from the students, because the proposed method had never been applied to any project, and the students were just about to learn the disciples and techniques of web engineering. There were other factors that might affect the result of the projects:

1. Time limitation
   The course was a two-month session. The students needed to learn various topics of web development beside the projects, and complete a solid web-based software system for the customers in eight weeks.

2. Students' background
   The students were either undergraduate or graduate. Only a few of them had taken software engineering courses before. Three of them had a full time job. None of them knew web engineering well.

3. Customer's needs
   As in most cases, the customers of both projects didn't have a complete list of requirements at the beginning. The students needed to help them to figure the requirements during the progress of the projects, especially for the KECSAC project.

The outcome was satisfactory. The students followed the process well and worked closely. At the end of the semester, they came out with good quality products and documents. Both the KECSAC office and the CS department were impressed with their work. The only missing part of the projects was testing. However, it seemed that there were only a few minor errors left after delivering the products to the customers, and they had been removed quickly.

5 Conclusion

A practice of UML for web development has been displayed in this paper. A process composed of several UML diagrams —use case, workflow, class, sequence, and activity—provides a short development term for constructing a web application. The process supports the very basic web system architecture, which consists of a server, applications, and web databases. Several guidelines have been provided in the paper for checking consistency among the diagrams. These guidelines help determine what UML diagrams to create next, how many to build, and what need to check in the new diagrams.

Because of its short development term, the process is easy to follow, and easy to move forward and backward for accommodating to requirement changes. At the end of the paper, two projects that use the proposed process to complete two web applications are introduced to show the usefulness of the method.

6 References

[1] Kuang-Nan Chang, "Consistency Checks on UML Diagrams," Proc. of the 2007 International Conference on Software Engineering Research and Practice, SERP'07, Las Vegas, June 25-28, 2007, pp. 242-246.

[2] David Day, "The Practicality of OO PHP," http://www.oreillynet.com/pub/a/php/2005/07/28/oo_php.html, July 28, 2005.

[3] Martin Fowler, *UML Distilled: A Brief Guide to the Standard Object Modeling Languages*, 3rd Edition, September, 2003.

[4] Gerti Kappel, et. Al. (Editors), *Web Engineering*, Wiley, Hoboken, NJ, USA, 2006.

[5] Craig R. McClanahan, "The Evolution of Web Application Architectures," the 7th Annual O'Reilly Open Source Convention, Portland, Oregon, August 1-5, 2005.

[6] Roger S. Pressman, and David Lowe, *Web Engineering: A Practioner's Approach*, McGraw-Hill, New York, New York, 2008.

[7] Edraw Soft, Edraw Flowchart—Workflow Diagram Software, http://www.edrawsoft.com/workflows.php, 2009.

[8] Christian Soltenborn, "Analysis of UML Workflow Diagrams with Dynamic Meta Modeling Techniques," Diploma Thesis, University of Paderborn, 2006

[9] Lijun Yu, Ribert B. France, and Indrakshi Ray, "Scenario-Based Static Analysis of UML Class Models," Lecture Notes in Computer Science, Springer Berlin/ Heidelberg, vol. 5301, pp. 234-248, September 22, 2008.

The Merging of Diverse Perspectives: Management, Customer, and Developer in an Iterative Development Planning Process

Robert F. Roggio and James R. Comer
School of Computing, University of North Florida, Jacksonville, FL 32224
Computer Science Department, Texas Christian University, Fort Worth, TX 76129

Abstract—*This paper addresses the unique views of planning undertaken by management, customers, and developers. The concept of an iteration is briefly presented, along with roles performed by management, customers, and developers in the iterative development process. All of these views map into a hierarchy of plans central to the successful culmination of the iterative project.*

Keywords: Iterative projects, management, customers, developers, assessment.

1. Introduction

One of the problems in managing an iterative project is the difficulty in evaluating and asserting 'progress' to the degree it satisfies managers, customers, developers, and other stakeholders having vested interests in the project's success. Management, customer representatives, and developers all have and need criteria to judge and track project progress. But different essential assessment parameters distinguish each of these diverse groups. Yet, at the same time, their assessments must map into a unifying successful project outcome that provides real business value to the client.

One fundamental issue may boil down to reconciling plans from the different constituencies into a set of unique yet unifying hierarchy of plans. Is there, then, some mechanism or common denominator that can be used to unite these stakeholders in such a way that seemingly unique plans are clearly threaded with common goals and not in conflict with other constituencies? According to some leaders in the field [1], the answer rests with iterative development. But the answer is not that simple - as this paper shall demonstrate.

2. The Iteration

The iterative approach forces developers to produce something useful and measurable as part of each iteration. This 'deliverable' (sometimes referred to as an executable deliverable) [5] can be tracked, managed, assessed, and reported to stakeholders. Is then the iterative approach the unifying technology that can be used to coalesce different perspectives in a development effort to form the basis for unifying planning efforts? To fully understand how it may be asserted that iterations underpin an all embracing planning scheme, one needs to first review three key components of an iteration. These are:

1) Production of the Deliverable
2) Continuous Risk Assessment
3) Overall Iteration Assessment

2.1 Production of the Deliverable

The notion of an iteration applies to many disciplines, and each adjusts the definition somewhat to fit the context of the application. According to *Wiki* [2] an iteration involves the act of "…repeating a process usually with the aim of approaching a desired goal or target or result. Each repetition of the process concludes with results that can be used as the starting point for the next iteration." Wiki goes on to offer a refinement of iteration in the context of project management: an iteration "…may refer to the technique of developing and delivering incremental components of business functionality, product development or process design. A single iteration results in one or more bite-sized but complete packages of project work that can perform some tangible business function. Multiple iterations recurse to create a fully integrated product." [2]

In our context, we view an iteration as a gradual enhancement of the solution definition with repeated core development activities. These may include requirements, design, coding, testing, and implementation. But to view an iteration as a simple mini-project greatly over simplifies the process.

First of all, when a specific iteration begins, the goals of the iteration must be totally understood and documented in an iteration-specific plan. Typically, until an iteration is about to begin, the precise activities to be undertaken might not be fully understood. This may be, to some degree, due to lack of completion of a previous iteration, which may not have successfully accomplished all objectives. Alternatively, it may, in fact, be due to newly-discovered changes. But in the spirit of time-boxed iterations, the previous iteration must stop on time and an objective assessment must be undertaken

to determine the degree of success of this effort. Part of the assessment may require some activities be moved into the next iteration (or a later one). Thus, the completion of one iteration may very well impact the plan for the next iteration. Nevertheless, when an iteration actually begins, the objectives of the iteration must be totally understood and the iteration plan must contain a set of well-defined objectives, evaluative criteria, and a well-defined process for objective assessment upon its conclusion.

Once a project is under way, the primary goal of an iteration must include construction of a measurable deliverable (increment) providing clear business value. This increment must be an integral part of the planning process and should be fully understood by all stakeholders at the outset of an iteration.

2.2 Continuous Risk Assessment

As we successfully produce an executable release (or increment) as part of the iteration deliverable, we must also undertake risk assessment. In addition to this increment, an assessment of risk is central to the iterative process. The consequences of risk assessment can impact an entire spectrum of activities ranging from the iteration plan for the next iteration, to the ultimate delay or cancelation of a project. Unforeseen technology issues, personnel problems, unanticipated financial problems, environmental changes, management turnover and other phenomena can be discovered at the conclusion of an iteration and may have an impact on successful project continuation. While the most consequential risk items are ideally addressed early in project iterations, continuous risk assessment requires a careful eye on newly discovered and lingering risk items and their potential or changing impacts.

It is common knowledge that, when using an iterative development scheme, high risk issues must be addressed and must be mitigated in order to proceed. In fact risk, in general, should be steadily reduced as development proceeds. Risk in early iterations may be focused on providing needed business value, market demands, financial considerations, evaluation of resources, proof of concept, time, schedule, environmental concerns and more. Iterations occurring later in the development effort more likely will center on assessing technology changes, newly identified/changed requirements, and team development issues. While acknowledging that each iteration can introduce additional risk – perhaps not previously recognized – the goal of assessing and mitigating risk is of prime importance. Failure to address such risk may lead to overall project failure.

2.3 Overall Iteration Assessment

One of the fundamental tenets of good iterative project management deals with iteration assessment. This activity, crucial to the project success, is central to determining if the project appears to be meeting its overall goals of providing the desired identified business value. Iteration assessment feeds the iteration plan for the next iteration, the development plan for the current evolution [3], and the overall project plan [1].

Assessment focuses on identifying the overall degree of project success in meeting the clearly defined goals of the iteration and must be objective in doing so. The developers can assess technology issues, and architecture, design, programming, integration, configurations and similar activities. In contrast, the customer representative, such as a business analyst, systems analyst and/or end-user, must assess the added business value provided by this increment, missing or additional functionality needed, and overall project progress. The customer, in the person of the business analyst or end-user, must assess the utility and usability of the increment. Management as part of the iterative process must objectively assess outcomes reported to them. Management assesses the success of the agreed-to iteration plan, the overall execution of the plan, and the objective assessment provided to them by the customer (business analysts and end users) and the developers. All of these must consider mitigation of risks and the discovering of new risks, if any.

3. The Merging of Diverse Perspectives

While there are many stakeholders in a non-trivial project, those arguably having the most impact on the overall success of the project are the developer, the customer, and management. Their project perspectives are quite different, yet they must, in the end, come together for validation.

3.1 The Developer

Life may be considered 'simple' for the developer. Within an iteration, the developer is concerned primarily with traditional activities such as analysis, design, and implementation (See Figure 1).

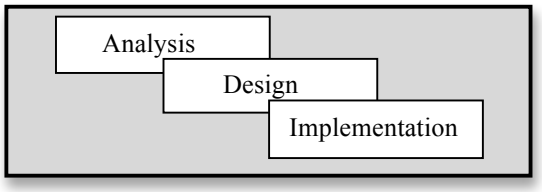


Figure 1 Developer Focus

Developers typically analyze the tasks at hand, design solutions, implement their designs via programming, undertake testing of various kinds and levels, and integrate added functionality into the evolving system. Developers deal with configuration management, development tools, change control, and the nuts and bolts of implementing a design based on a solid architecture. Developers are typically not terribly concerned with the return on investment, benefits

realization, and risk management. Within a time-boxed iteration, activities need to be well-defined, objectives clear, and behaviors conducted in a stable development environment supported by modern, established software tools. Developers appreciate small projects – design, code, test, integrate added functionality – and happily work toward the measurable objectives at the conclusion of the iteration to include production of an executable release (internal or external) that provides a measurable increment of added business value.

3.2 The Customer

Customers can be many and wear different hats. But the role of the business analysts and the end-users – stakeholders who desire that the evolving product meet the requirements in functionality, accuracy, usability, and utility – is essential. One key responsibility of the customer representation is how to interface with developers to offer objective feedback in support of the next increment. According to Bittner and Spence [1], it is critical to understand that much of the objective feedback rising to senior level management originates more from customers and definitely less from developers. It is only through this kind of objective customer feedback that senior level management achieves a comfort level that the development efforts of all stakeholders will converge to an ultimate business solution.

The customer representative role, then, is essential for iteration success, as the customer can provide a confidence builder to developers that the latter are indeed participating in the evolution of an application with real business value. The customer representative further may interact with management providing positive assessment that overall development is proceeding as planned. Recognizing that management may be quite isolated from analysis, design, and programming details, assurance of on-time evolving business value, continuous risk assessment, and a satisfied customer-base will pay huge dividends in overall project success.

As noted, while the developer is nominally interested in analysis, design, and implementation, the customer representative superimposes a requirements activity on the 'front end' of an iteration, and a 'system test' activity on the back end of the integration that culminates in a tested release. [1] (See Figure 2) From the customer perspective, the requirements will encompass the specific objectives of the iteration to include new functionality or change requests. These specifications must precede analysis. The customer representative must always work to ensure that a minimal set of requirements needed to solve the business problem are identified and addressed. More features are not necessarily better. History is replete with failed projects having an overwhelming number of requirements - many of which were simply not needed.

From the customer perspective, the iterative process is viewed as developing a "build" for some requirements and/or change requests, demonstrating the release, inspection and acceptance, and a repeatable process. [1] As important as the business analysis is in ensuring the evolving system meets customer needs, it is equally vital that customer acceptance includes the end-user. Only by including the end-user in the iteration assessment activity can basic key features such as the utility of the application (does the interface support the day-to-day workplace activities?), the learnability (is the application and its interface easy to learn and are there 'shortcuts?), and the usability (is it easy and intuitive to use?) be objectively assessed. [4]

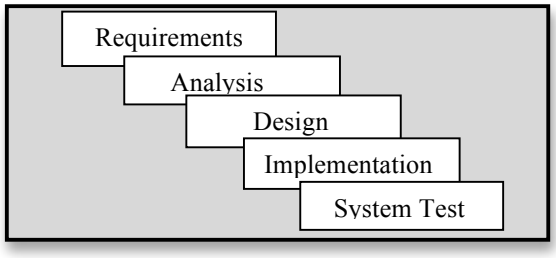


Figure 2 Customer Focus

3.3 The Management

Management exists at all levels to a greater or lesser degree in a development environment. So, to scale 'management' in this context, we will take the Bittner and Spence approach and restrict management to include the project manager and the quality assurance manager.

Management's concerns understandably reflect high level issues such as, is the overall development effort really solving the right problem; will the desired outcomes provide clear business value delivered on time and within budget with resources provided; are the activities of the development effort clearly moving toward the sought-after business solution; and is risk continually addressed and mitigated as increments are delivered. Management considers a project in the enterprise sense and, thus, as a series of one or more evolutions each of which delivers major increments of business value.

Unlike the developers who view each iteration as hunks of analysis, design, and implementation, and the customer who bound this set of activities with requirements on the front end and system testing on the back end (Figures 1 and 2), managers agree on the overall objectives of the iteration, the evaluation criteria to be used to determine success of the iteration, and exactly how the iteration is assessed to determine the project is moving toward a successful conclusion. Once the agreements are established, the iteration plan is executed by the developers in concert with the customers, and a comprehensive assessment ultimately takes place which compares the outcomes of the iteration to the objectives and evaluation criteria. Assessment results are then used to advance the product and project (hopefully) and fold into subsequent iteration plans.

This Agree-Execute-Assess process is the management cycle. (See Figure 3) No longer does management measure progress by the completion of huge design documents, many technical and/or managerial reviews, and the production of lines of code. Management now focuses on produced software. [5] Via the objective assessment provided by the customer to management and management's own assessment occurring at the end of iterations, this modern role can better control the project. Management looks for coded, measured, tested, verified, and integrated work products satisfying the customer that clearly indicates the project is progressing on time and within budget.

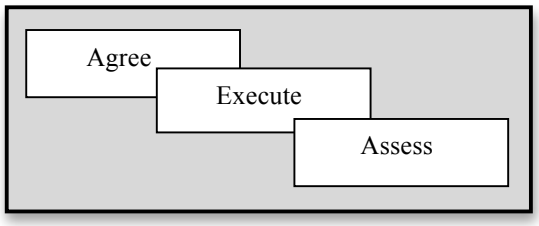


Figure 3 Manager Focus

4. Iterative Projects Plans and Related Work

So the concerns of management, the customer, and the developer in the iterative project are reasonably clear. Perspectives are one thing. But in the context of iterative planning with very diverse views of stakeholders, can these concerns be mapped into meaningful plans and planning documents for management, customer, and developer that come together and support a cohesive, trackable development effort?

Insofar as planning is concerned, there is no single issue and certainly no single limitation to control. In a way, a muddling through approach where we solve one problem at a time and use that knowledge to solve the next problem (see piecemeal engineering) [7] can be used to frame some of the uncertainties in software development. Barstad [8] discusses muddling through as a theory for continuous adaption, which is often exactly what we are doing as we develop modern systems. The principle is to take many "small steps in an adaptive and learning process instead of making larger and more fundamental changes."[8] Barstad goes on to assert that this muddling through model can be used to support the approach to satisfying a long-term goal. The tactics suggested include developing long-term goals in a strategic plan, adapted through administrative planning, and accomplished via daily operative work.. Lastly, he goes on to say that this model does depend upon a centralized management and but leaves the adaptation to subordinate levels. This seems to imply a hierarchy or 'pecking order' of plans. It also assumes that the principals at each level have the same kind of power – a fight between equals. This suggests separate but equal, meaningful plans.

Planning and accommodating are complementary activities. Planning is a very complex undertaking that includes identifying goals to be realized by implementation, a separate exercise, which is not usually considered part of the plan. Planning in most contexts is a continuous activity and is never finished. [9] Consider planning in an Agile development process.

In Agile, our plans can change (and do) as working software is delivered. As a project unfolds, the plan changes. According to Wells [10], "Some Agile projects don't even have iteration ends. They remain Agile by balancing the need for stable requirements with the need to change requirements while launching working software on a regular and frequent basis. Iterations are just an easy way to demarcate when changes are accepted, when a new plan is create, and when working software is released to customers. Shorter iterations give more opportunities to plan.." But even Agile typically has a series of plans and Wells cites release planning (groups of stories selected whose features are to be developed and released as a unit), the iteration or sprint plan (stories to be addressed in the next iteration), and the daily plan, often referred to as the stand-up meeting, where the detailed individual plans of the day are articulated. It is the iteration plan where the developer, manager, and customer meld features (often captured in use-cases), time, amount of work into this iteration. Here again, we have a series of plans that must co-exist and equally share in the overall planning process.

To understand how the iteration can supply the unifying technology and thus underpin or drive a successful project across plans, one may also use the Unified Process [3] as a framework. In the Unified Process one or more iterations constitute each phase in an evolution below, where the evolution concludes with the major deployable release. It is interesting to note that many projects are concluded as a single evolution, and customer service, training, installation, conversion, end-user support and other activities now provide continuous customer support.

As it turns out, many modern major projects developed under the Unified Process deliver a system in a series of these evolutions and consequently (for larger projects) often provide essential core and critically-needed functionality up front in early evolutions with added, maybe discovered, and sometimes lesser functionality delivered as part of a subsequent evolutions. (This may happen when major requirements in the original evolution are somehow missed and discovered simply too late to be incorporated into the current evolution and still meet time-to-market and other concerns) Additional evolutions may be delayed, spaced out, or overlapped in time. Each evolution, however, goes through all the phases in Figure 4 and each deploys major business value.

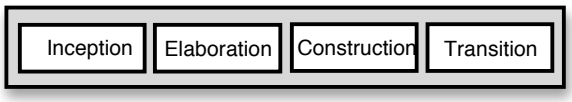


Figure 4 An Evolution

An overall major project may well consist of a series of project evolutions as shown in Figure 5. Given this backdrop, history has shown that plan-driven processes, such as the traditional waterfall model, rapidly become unwieldy and very difficult to manage as change is discovered. [5] Sometimes changes are so plentiful in number and significant in impact that they may not be incorporated into an existing (and base-lined) plan; thus, jeopardizing the project. Using this model, these issues may result in a major unplanned follow-on development effort. Even when using an iterative model with a high probability of success for initial activities, repeatedly multiplying probabilistic impacts of discovered changes embraced within this model will yield greater and greater inaccuracies. Yet, planning at all levels is needed to coordinate overall work efforts and to assure all stakeholders that the project is proceeding successfully. Consider this approach.

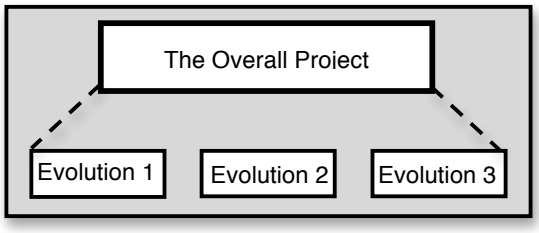


Figure 5 The Overall Project

- **Overall Project Plan.** Management needs dictate that the project plan be simple, brief, and contain the anticipated number of evolutions. Typically the overall plan will contain milestone dates, functionality delivered (prioritized), high-level risks to be addressed, and resources needed. Bittner and Spence assert that an entire project cannot be planned in any great detail in the beginning before any iterations take place. It is unlikely enough knowledge about the project is fully understood. To be effective, an overall plan should thus be planned with common sense (sometimes ignored), be lightweight and summarize long-term directions for the project with the major releases identified that provide high level, clear business benefits. While the plan at this level is, of necessity, simple and focuses primarily on major releases (in terms of evolutions, if appropriate), the overall plan provides an umbrella structure for lower level plans so that the lower level plans can be in synch with the overall plan. Thus, the lower level development plans must map into or be subsumed by the overall project plan.

- **The Development and Iteration Plans** In practice, major inputs to this overall project strategy come from the development plan, which is the plan for the current evolution. What constitutes the major release? What is the (first) major thrust of the effort? This plan, of necessity, includes operational milestones associated with the end of phase milestones (Inception, Elaboration, Construction, and Transition). Central to this development plan is the strategy for each phase within an evolution to include the projected number and purpose of the iterations contained (but not too much granularity). At this stage, these iterations in the phases are skeletal at best and may be taken from high level use cases, where major descriptions of functionality are found. Thus, plans here are initial descriptions of what is to be done driven by the requirements. See Figure 6.

Bittner and Spence point out that the overall plan tends to be forward looking, low fidelity, low precision, visible to the organization, and optimistic whereas (as we ratchet down) the development plan provides the plan for the evolution and similarly consists of detailed iteration plans within each phase that have high fidelity, high precision, high visibility and are somewhat pessimistic. [1] In marked contrast to the overall plan, the ultimate details are thus pushed down to the working level in terms of the detailed iteration plans.

Figure 6 A Hierarchy of Plans

5. Putting it all together

The merging of these perspectives and these plans into a composite, integrated model is an interesting undertaking. First, software development projects may be considered 'complex adaptive systems.' [5] These systems, by definition, define those where the number or types of components in the system and the number and nature of the relationships among the parts are 'non-trivial.' [5] If this is true, then it seems reasonable that attempts to determine an integrated set of plans that address the merging of non-trivial parts related in complex ways must also be rather complex. The agile approach uses a series of plans; Barstad [8] refers to a muddling approach, which makes great sense and reflects many development efforts. . In his discussion of a project's organization, Mark Kennaley [5] points out that it is difficult if not impossible to fully understand at a system level all the details in the white box sense. But, he goes on to cite that treating the project organization (and, we infer, the planning of the project) "as a very complex black-box of dynamism is all that is needed in aggregate, as long as we can observe the results or output from that very busy network of self-organizing agents." The authors of this paper suggest a another prototype of integrated plans (see Figure 7).

The basic plans are in the center in rectangles. Major focus of each is included. Major stakeholders appear on the left. Cardinality (many to n to 1) Iteration Plans are subsumed into Development Plans which are subsumed into the Overall Plan.

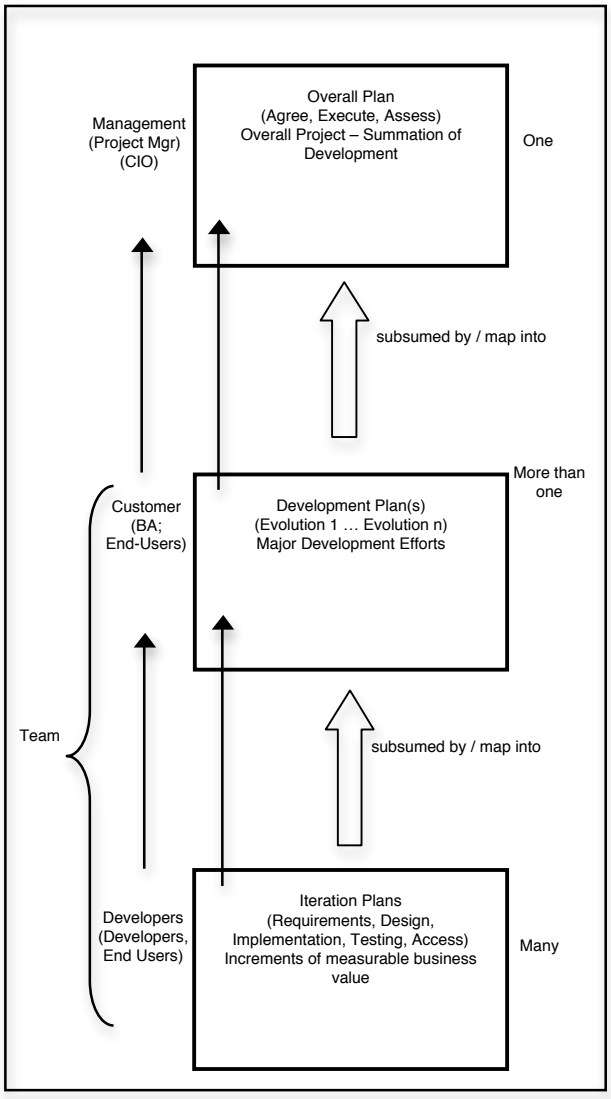


Figure 7. A Hierarchy of Plans: Management, Development, Iteration

6. Conclusion

Iterative development can underpin the work accommodated within an iteration, evolution, and project. Individual iterations provide increments of business value; iterations spanning an evolution deliver the major releases; and the series of major releases provides the ultimate business value for a major project. Concomitantly, low level detailed iteration plans map into the development plan for the evolution, which, in turn, maps into the overall project plan. Management, customers, and developers all take different yet complementary views of an evolving system and each have different yet complementary plans for success. Successful projects promote, respect, and encourage these differences. All are pursuing the same ultimate success.

7. References

[1] Kurt Bittner and Ian Spence, "Managing Iterative Software Development Projects," Addison-Wesley, 2007, ISBN: 0-321-26889-X

[2] *Wikipedia*, Definition of 'iteration'.

[3] Philippe Kruchten, "The Rational Unified Process – An Introduction," 3rd edition, Addison-Wesley, 2004, ISBN: 0-321-19770-4

[4] Timothy Lethbridge and Robert Langaniere "Object-Oriented Software Engineering," McGraw-Hill, ISBN: 0072834951, Aug 2001

[5] Mark Kennaley, "SDLC 3.0 Beyond a Tacit Understanding of Agile," Fourth Medium Consulting Inc., 2010, ISBN 9780986519406

[6] Walker Royce, "Software Project Management – A Unified Framework," Addison-Wesley, 1998, ISBN 0-201-30958-0

[7] K. R. Popper, "The Poverty of Historicism" Harper Torchbooks, New York: Harper & Row, 1964

[8] Johan Barstad, "Iterative Planning Processes; Supporting and Impeding Factors," Volda, Norway, *2002*. http://www.metla.fi/eu/cost/e19/barstad.pdf

[9] "Guidelines for Land-Use Planning," FAO, Rome, Italy, *Overview of the Planning Process*, ISBN 92-5-103282-3, reprinted 1996.

[10] Don Wells, "Iterative Planning," http://www.agile-process.org/iterative.html,

Capturing Dynamic Behavior in Relational Model

A. Tulika Narang[1], **B. Dharmendra K. Yadav**[2]
[1],[2]Computer Science Engg. Department, MNNIT, Allahabad, India

Abstract— *The schema representation of relational model does not define the dynamic behavior of the domain being modeled. The change in state of database application is not represented by schema. This limitation can be overcome by defining a metapattern that includes the state parameter to represent the dynamic behavior. The paper is an attempt to resolve this issue as a solution. The metapattern represents the relational database at metalevel. The objective of the paper has been achieved by merging ontology and object oriented methodology to design a metapattern for relational database. Metapattern is a framework that is applicable to various application domains. It is in form of tuple and describes the application domain at meta-level.*

Keywords: Metapattern, Ontology, Object modeling, Relational Model

1. Introduction

The schema of relational model describes the real world domain. The data base schema describes the database. Schemas are represented using schema diagrams. The schema diagram displays only static aspects, relation name and attributes. The dynamic behaviour due to change of state is not represented in the schema diagram. The constraint such as a book can only be issued if it is on the rack cannot be represented in schema diagram. Similar constraints related to state change are not described using the schema diagram. The paper is an attempt to solve this issue. The focus is to merge ontologically derived concepts and object oriented constructs to build a metapattern for the relational model. The ontological and object oriented constructs such as objects in real world, their associated properties and behavior, events and state change due to events is combined to build a metapattern for a relational model. Metapattern describes the real world domain and comprises of variables to represent the domain under consideration. Object and associated attributes represent static semantics and state change due to occurrence of an event describes the dynamic behaviour [15]. Although the schema describes the various entities and their related properties but do not describe the dynamic behavior. An object changes state due to an event which is not represented by the schema. The paper is an attempt to overcome this problem by including a time variable in the metapattern representation.

Metapatterns are patterns of patterns [11]. Metapattern determines objects that support useful roles in a specific context and represents the domain at meta-level. Metapattern is a powerful abstraction technique to large-scale information analysis and modelling. It structurally incorporates the elements of both time and multiple contexts. The fundamental concepts of metapattern are context and intext, alongwith type, time, and compositions. In particular, metapattern deals with recognition of multiple contexts results in a powerful approach to conceptual information modeling.

An approach has been taken which merges ontology and object oriented constructs for building a metapattern for relational model. In computer and information science, an ontology is a formal representation of knowledge as a set of concepts within a domain, and the relationships between those concepts [13]. It is used to reason about the entities within that domain, and may be used to describe the domain. The term ontology has its origin in philosophy, and has different meaning in different contexts. The core meaning within computer science is a model for describing the world that consists of a set of types, properties, and relationship types. Historically, ontologies arise out of the branch of philosophy known as metaphysics, which deals with the nature of reality of existence. This fundamental branch is concerned with analyzing various types or modes of existence, often with special attention to the relations between particulars and universals, between intrinsic and extrinsic properties, and between essence and existence.

An ontology of object-oriented systems can be represented as class, instances of class, relation between class and subclass. Object oriented paradigm represent the real world as objects. Objects with similar characteristics and behavior are grouped to form classes. They communicate with each other via message passing or shared memory. Any two objects are distinct and distinguishable in the real world. They represent real world entities and are a means for encapsulating abstractions. They have attributes associated with them. Each attribute represents a particular property of the object. Attributes have values that are prone to change with respect to time. They exhibit dynamic behavior features by the combination of its attributes and values. An object is *an entity with a well-defined boundary and identity that encapsulates state and behavior* [10].

2. Background and Related Work

Everything in the real world changes state with respect to time. Time signifies state change [1]. Thus representing temporal aspect is essential for representing a real world domain. To capture time a framework is required. The framework can be represented as a metapattern which includes time variable alongwith context and intext [11].
The concept of merging and defining mapping rules to build the metapattern has been presented in various literatures [2,4,8]. The metapattern has been defined by merging ontology and object oriented paradigm. The merging of ontology and object concepts is defined on the basis of of certain mapping rules between ontologically derived concepts and object oriented language constructs [4]. The word "ontology" has a long history in philosophy, in which it refers to the subject of existence. In the context of knowledge sharing Gruber defined ontology, *a specification of a conceptualization*. That is, an ontology is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. An ontology provides a shared vocabulary, which can be used to model a domain that is, the type of objects and/or concepts that exist, and their properties and relations. An ontology defines a set of representational primitives with which to model a domain of knowledge or discourse. The representational primitives are typically classes (or sets), attributes (or properties), and relationships (or relations among class members). The definitions of the representational primitives include information about their meaning and constraints on their logically consistent application. In the context of database systems, ontology can be viewed as a level of abstraction of data models, analogous to hierarchical and relational models, but intended for modeling knowledge about individuals, their attributes, and their relationships to other individuals. Common components of ontologies include [3,4]:

- *Individuals* instances or objects (the basic or "ground level" objects)
- *Classes* sets, collections, concepts, classes in programming, types of objects, or kinds of things
- *Attributes* aspects, properties, features, characteristics, or parameters that objects (and classes) can have
- *Relations* ways in which classes and individuals can be related to one another
- *Function terms* complex structures formed from certain relations that can be used in place of an individual term in a statement
- *Restrictions* formally stated descriptions of what must be true in order for some assertion to be accepted as input
- *Rules* statements in the form of an if-then (antecedent-consequent) sentence that describe the logical inferences that can be drawn from an assertion in a particular form
- *Axioms* assertions (including rules) in a logical form that together comprise the overall theory that the ontology describes in its domain of application. This definition differs from that of "axioms" in generative grammar and formal logic. In those disciplines, axioms include only statements asserted as a priori knowledge. As used here, "axioms" also include the theory derived from axiomatic statements.
- *Events* the changing of attributes or relation

Ontology is concerned with the general features and facts about the real world. Ontology focuses on what kind of objects are found in the real world and how these objects are organised [8]. Ontologies have several advantages such as they promote and facilitate interoperabilty among information systems and sharing and reuse of knowledge among systems. They provide a shared and common understanding of a domain that can be communicated between people and application systems. Ontology is an explicit formal specification of how to represent the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. Ontology describes features and facts about the real world. Ontology is a conceptual model that is used to represent a particular domain [5].

3. Mapping from Ontological representation to Object Model

The paper attempts to merge the ontological and object oriented concepts to build a metapattern. The merging process is done by defining a one-to-one correspondence from ontological concepts of the real world to object oriented constructs [7,13].

3.1 The Bunge-Wand-Weber model

Bunges ontology serves as a foundation for Bunge-Wand-Weber model(BWW) ontology [9,13]. Bunges original ontology is considered as a general system theory. Wand and Weber have adapted it to model information systems. Their ontology is deemed to be sufficient and necessary to describe every aspect of a modeling language.

BWW ontology is a generic framework for analysis and conceptualization of objects. The domain of Bunge's ontology is the physical world. The world comprises of objects that exist in time and space such as people, buildings, tables. Objects have properties or characteristics such as dimensions, intelligence, mass, and density. The role that ontology plays in conceptual modeling is to guide the modeling process with a particular philosophy of how to describe the domain of discourse. For example, an ontology that defines things as material objects. Some constructs of the BWW ontology [2,6]:

- *Thing* The world is made of things that have properties. It is an elementary unit in ontological model. It is a substantial object having existence in reality. The real world is made of things.They possess properties and are known in real world via properties.
- *Composite Thing* A composite thing may be made up of other things (composite or primitive).
- *Conceivable State* The set of all states that the thing may ever assume.
- *Transformation of a Thing* A mapping from a domain comprising states to a co-domain comprising states.
- *Stable State of a Thing* A state in which a thing, subsystem or system will remain unless forced to change by virtue of the action of a thing in the environment (an external event). The notion of a state is based on the postulate that every thing is in some state or the other at a given time.
- *Property* We know about things in the world via their properties. A property is modeled via function that maps the thing into some value. A *mutual* property is meaningful only in the context of two or more things.
- *System* A set of things will be called a system.
- *Subsystem* A system whose composition and structure are subsets of the composition and structure of another system.

3.2 Mapping from ontological concepts to object oriented concepts

The following rules has been used for mapping:

- Rule1: Things are mapped to objects.
- Rule2: Properties of things are mapped to object attributes.
- Rule3: Methods and operations change the values of attributes. State represents the current values held by various attributes of objects. The state of a thing is the aggregate of its properties at a point in time. Hence it can be represented by the aggregate of those attributes that represent its properties. An event causes the change in state of a thing [3,4].

4. Metapattern for specification of dynamic behavior for relational database

A database based on relational model comprises of tuples. Each tuple is a separate object with certain properties. The change in the values of attributes results in the change of database state. The metapattern comprises of three main components [11]:

- Context
- Intext
- Time

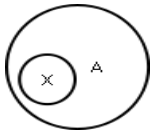


Fig. 1: Object X belongs to context A

Context is an ordered collection of objects and relationships. An object behaves according to a particular context. It belongs to a particular context.
An object does not have a primary singular identity. Instead an overall identity is the collection of identities determined by the context. An object belongs to a particular context. The behavior of an object is dertermined by the context Fig.1. A specific object in a specific context behaves in a unique way [11].

The relational model comprises of tables and each table has multiple tuples [14]. Each table in a relational model represents a context. The tuples of the relation are distinct and distinguishable objects belonging to the same context. *Context* specfies the properties that correspond to a specific type. It represents the various attributes that are valid within a particular context. An object that exhibits all the properties of a type is qualified to be of that particular type. A set of properties that corresponds to a specific context is called *Intext*. The columns in the relational model represent the attributes of the particular context. Thus the intext defines the properties of objects within the particular context.
Time is defined on the notion of state change. The state of the database changes with the change of values of the attributes. The concept of a state is based on the notion that every thing is in one of the states at a particular instance of time [1].
A metapattern for a relational database is presented here. The schema representation of relational model only focuses on static semantics. A schema is represented using a schema diagram [15]. A schema diagram displays only certain aspects of a schema such as relation name, associated attributes. A schema is also called the intension. It does not describe the dynamic behavior. An object in the real world changes state due to occurrence of an event. Event represents an external stimulus that causes the change in value of the attribute. This constraint is being overcome by including a time variable in metapattern representation [1].
The metapattern describes the relational model and is represented in form of a tuple comprising of three variables. The three variables capture context, intext and time [11].
Mp = $\langle$ Co, Io ,T $\rangle$

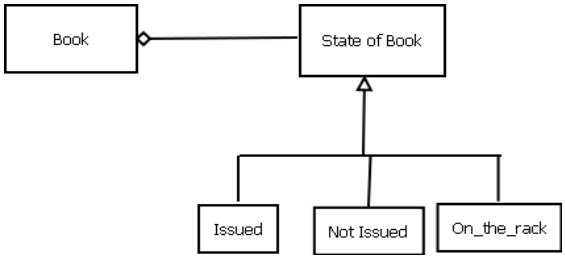


Fig. 2: Representing book context using state pattern

where **Mp:** Metapattern for relational model
Co: Context,
Io: Intext,
T: Time

The suffix o represents a particular object. Context according to the metapattern is a variable to be valued within conceptual information models. A real world scenario can support a number of contexts or context instances. A context represents a particular type. Intext represents properties of context. The metapattern represents a system at the metalevel. It constructs a framework independent of specfic domain. Metapatterns are an elegant and powerful approach to categorize and describe a design pattern at metalevel [13]. The metapattern captures the time domain constraint by including time variable in metapattern tuple. The metapattern represents the domain at the most abstract level and maps to a design pattern [12]. A design pattern is an instance of mettapattern. It is like a template that can be applied in many different situations. The selected pattern is a behavioral design pattern, *State pattern*. The design pattern is applied to a paticular domain [15].

The state pattern is choosen because it represents the state change and the behavior of the system changes by swithching between a set of operations. A behavior of a system is a mapping b:T $\rightarrow$ S. T represents the temporal domain and S is the State space [1]. The behavior represents the system's state (i.e. value of its elements) in various time instants of T. The pattern allows an object to alter its behavior when the internal state changes. The object will appear to change its class. The behavior of an object depends on the state. It changes its behavior at run time depending on that state [16].

In library management system, Fig.2 Context: Book as a Library Item
Intext: Title, Author, Price, ISBN, Publisher
Time: ontheRack, issued, not issue

5. Conclusion

The paper is an attempt to solve the constraint of relational schema by including time variable in the metapattern representation. By including time variable the dynamic behavior associated with the real world can be explicitly represnted. An approach of merging ontology and object oriented concepts has been taken to bulid a metapattern. Ontology and object oriented constructs closely represent the real world. A one-to-one mapping from ontology to object oriented paradigm can be easily defined for the reprsentation of metapattern. With the metapattern reresentation an object of the real world can be unambiguously represented.

References

[1] Carlo A. Furia, Dino Mandroli, Angelo Morzenti and Matteo Rossi,*Modeling Time in Computing: A Taxonomy and a Comparative Survey*, ACM Computing Surveys, Vol 42, No. 2, Article 6, Feb 2010.
[2] Dr. Waralak V. Siricharoen,*Ontology Modeling and Object Modeling in Software Engineering*,International Journal of Software Engineering and its Applications, Vol 3., No.1, Jan 2009.
[3] Waralak V. Siricharoen,*A Software Engineering Approach to Comparing Ontology Modeling with Object Modeling*,International Symposium on Computer Science and its Applications, 2008
[4] Joerg Evermann and Yair Wand,*Ontology based object-oriented domain modelling:fundamental concepts*, Springer-Verlag London, Jan 2005.
[5] Islay Davies,Peter Green,Simon Milton and Michael Rosemann,*Analysing and Comparing Ontologies with Metamodels*, 2005
[6] Joerg Evermann and Yair Wand,*Ontological Modeling Rules for UML:An empirical aassessment*, The Journal of Computer Information Systems,2006
[7] Michael Rosemann and Boris Wyssusek,*Enhancing the Expressiveness of Bunge-Wand Weber Ontology*, Proceedings of Eleventh Americas Conference on Information systems, 2005
[8] Boris Wyssusek,*On Ontological Fondations of Conceptual Modelling*, Scandinavian Journal of Information Systems, 2006
[9] Arvind W. Kiwelekar and Rushikesh K. Joshi,*An Object Oriented Metamodel for Bunge-Wand-weber*, In Proceeding of SWeCKa 2007, Workshop on Semantic Web for Collaborative Knowlege Acquistion at IJCAI,Jan 2007
[10] Gregor Engels,Reibo Heckel and Stefan Sauer,*UML:A Universal Modeling Language*Springer-Verlag Berlin Heidelberg 2000
[11] Pieter Wisse, Metapattern: Context and Time in Information Models, Addison Wesely, ISBN 0-201-70457-9, 1st edition, August 1994
[12] Wolfang Pree, Design patterns for Object Oriented Software Development, Addison Wesely, ISBN 0-201-42294-8, 1st edition, December 2000
[13] Mario Bunge, Treatise on Basic Philosophy, Vol 3., Ontology I:The Furniture of The World, Springer, ISBN 9027707804, 1st edition, June 1977
[14] Elmasari,Navathe, Fundamentals of Database Concepts, Pearson Education, ISBN 978-81-317-1625-0, Third impression 2009
[15] Craig Larman, Applying UML and Patterns-An Introduction to Object Oriented Analysis and Design and Iterative Development, Pearson Education, ISBN 978-81-7758-979-5, Fifth impression 2008
[16] Erich Gamma, Richard Helm, Raplph Johnson, John Vlissiles, Design Patterns-Elements of Reusable Object Oriented Software, Addision Wesely, ISBN-0201633612, 1st edition 1994

Towards a Shared Platform for Virtual Collaboration Analysis

Thomas Kowark, Matthias Uflacker, and Alexander Zeier
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
{firstname.lastname}@hpi.uni-potsdam.de

Abstract— *Prior applications of a system to monitor IT-mediated communication activities of engineering teams provided new insights into the collaboration behavior during the early phases of engineering projects. Building on those findings, we now present an architecture for a platform that aims to establish 'out-of-the-box' monitoring capabilities for virtual team environments and to facilitate the sharing and evaluation of recorded activities within a larger research community. To demonstrate the applicability of our instrument, we present the intended usage workflow and provide an outline of current applications in the domains of software engineering and engineering design. Our vision is a common service for capturing and analyzing virtual collaboration activities that promotes comparative research and team diagnostics across engineering disciplines.*

Keywords: Virtual Team Collaboration, E-Research, Software-as-a-Service, Semantic Web

1. Introduction

Virtual collaboration has become an integral part of the daily work of engineers. Information sharing, communication, and coordination activities carried out through email, groupware, and online services increasingly determine the way how engineering teams design and prototype new products, software, or services. With the critical importance of effective and efficient team communication being generally acknowledged [3], [6], the question of how virtual collaboration behavior affects the quality of engineering processes still remains largely unanswered. One reason for this lack of understanding is the difficulty to systematically observe, analyze, and compare such processes. While recent research has begun to examine virtual collaboration in co-located and distributed design teams more closely, the lack of generally applicable instruments for monitoring the broad range of online team activities hinders in-depth investigations.

Existing approaches commonly rely on tailor-made tools that work well in a specific work scenario or with certain collaboration tools, but are not applicable in different observation contexts. Transferring those instruments to other collaboration scenarios is often impossible or involves rewriting of the software or interfering with the process under study. Repetitive efforts and high costs of implementing customized solutions to observe virtual collaboration processes are the result. Furthermore, the use of isolated instruments and data formats prevents other researchers to replicate or verify previous findings, which is an important criterion for relevance and rigor in empirical design research [4]. A broadly applicable technological foundation for monitoring and studying virtual collaboration in the field is needed. It has to minimize the efforts for data collection and analysis and facilitate comparative research.

We have developed *d.store*, a customizable service platform to collect and analyze virtual collaboration activities during project runtime and in a non-interfering manner [17]. The platform is configurable and can be utilized to capture collaboration activities from heterogeneous groupware systems, generating a single record of temporal and semantic relationships between identified actors and resources. A service interface provides the functionality that is needed to explore trends and to analyze detailed characteristics in the collaboration behavior of the observed teams. First applications of the instrument in the conceptual design phases of eleven small-group distributed engineering design teams have demonstrated the feasibility of our approach as well as collaboration metrics that correlate with the performance of the teams [16], [15].

In order to make the *d.store* services and, hence, the data that has been collected during observations available to a larger research community, our next iteration of the platform seeks to incorporate functionality that facilitates its integration and application in 3rd party project environments and which allows for easy sharing of anonymized activity records. In this paper, we introduce the targeted system architecture and motivate a common approach to virtual collaboration monitoring. First, a brief overview of related work is given to highlight the contributions of the system for the field of virtual collaboration research. Then, we present the architecture of the platform along with an outline of the intended usage workflow. The paper concludes with an introduction of example applications of the system and the identification of possible future research activities.

We believe that this approach will stimulate relevant and rigorous findings in empirical engineering research and help

scientists and practitioners to understand relevant factors in virtual team collaboration.

2. Related Work

Monitoring and analyzing digital collaboration activities to deduce beneficial or detrimental developments within project teams has been subject to prior research.

An approach that also aggregates data from different data sources was developed by Ohira et al. [9]. The 'Empirical Project Monitor' relies on numerous feeder applications that parse data sources like source code management systems, bug trackers, or email archives. It provides a number of preset visualizations for this data. Additionally, an underlying communication model tries to detect flaws within the collaboration behavior based on previous empirical studies. The main difference between our platform and this approach is the creation process of the communication model used for deviation detection. Instead of being generated from an ever increasing database of collaboration data, it is statically implemented within the application.

Wu et al. created a metric-based, multi-agent system that supports project management by gathering information from various sources of the development environment and deducing the current project status by means of intelligent agents that relate software development activities to the previously created project plan [14]. While the system is equally applicable to various project setups, the different installations remain isolated from one another. Thus, insights about activities that have beneficial or detrimental effects cannot be made available to other projects without updating the evaluation logic of the software agents and redeploying the system.

Reiner [12] presented a proposal for a knowledge modeling framework that supports the collaboration of design teams. Communication information has been deduced from explicit interactions between members of a design team by a software tool that he developed to provide a prototypical implementation for the proposed framework. This research laid the early foundation for our work but did not address generalization of the findings made during single projects by validating them in an automated manner against other gathered data.

Microsoft's Team Foundation Server [8] is a commercial collaboration tool suite that allows its users to analyze and compare the collaboration behavior of the teams using this platform. It is, however, limited to collaboration activities that are performed using one of the provided tools. If, for example, different version control, bug tracking, or email systems are used, the corresponding activities cannot be analyzed.

E-research platforms are an equally interesting field of related work. Approaches like the one presented by Abidi et al. [1] enable researchers to jointly work on research topics by distributing data collection and evaluation activities. Since all involved persons are using a shared system, collected data and evaluation results are available to each of them. However, such a system was, to our knowledge, not developed and applied in the area of virtual team collaboration analysis.

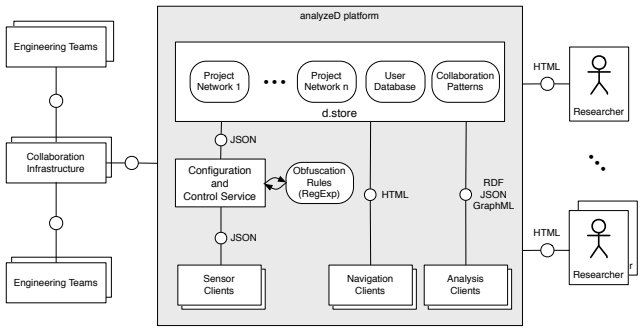


Fig. 1: **Basic architecture of the proposed platform.**

3. Platform Architecture

The goal of our research is to answer the question: How to enable researchers to easily compare virtual collaboration in different project settings and share the data and analysis results?

As an approach to a solution, we have created a shared platform for the collection, analysis, and sharing of digital collaboration data and results from their analysis. The basic building blocks of the platform are depicted in Figure 1.

The system enables researchers to use a set of readily accessible services to convert and upload the collaboration traces of the projects they want to analyze. This data can, in turn, be scrutinized through a set of analysis clients While those use-cases were also possible with the aforementioned d.store platform, the newly created system also builds up a database of projects that can be used for verifying the conclusions drawn within single project analysis. Furthermore, the system allows the sharing of analysis results in a way, that the occurrence of certain collaboration patterns can be automatically detected in newly uploaded collaboration networks.

In the following, we describe the building blocks of the system architecture by outlining the intended usageÂt' workflow and highlighting the services that implement the respective functionality.

3.1 Data Collection

The initial step of platform usage is data collection. The d.store project introduced the notion of sensor clients to provide unobtrusive and reusable services that are able to capture collaboration events from heterogenous groupware

tools. Though each client is specifically implemented to parse a certain groupware tool, all clients produce the same output format.

As an addition to this concept, we introduced a central 'Configuration and Control Service' that serves as a repository for sensor clients. Hence, users of the platform get an overview of available sensor clients and can choose the ones appropriate for their specific project setups. Each client has to implement a REST-interface that allows the control service to invoke the sensors in a generic manner. One endpoint of the interface returns the required input parameters for the sensor clients. This enables the automatic creation of web forms for entering access credentials for the tools under investigations. By that, the platform can be used to parse the digital collaboration data of entire projects by performing two simple steps - choosing the appropriate services and entering the access credentials.

A basic set of sensor clients that covers two different email archive formats, three types of wiki solutions, two source code management systems, and two bug tracking systems has been implemented, thus far. As new groupware tools emerge or existing ones are being updated, however, new sensors need to be implemented. Those new sensors have to implement the given REST-interface and can be made available to all users of the platform. Thus, duplication of effort is reduced, as only one sensor client implementation per groupware tool is necessary.

3.2 Creation of Collaboration Networks

As previously mentioned, all sensor clients have to transform the captured collaboration events into a standardized output format. This output, which is generally encoded by means of the Javascript Simple Object Notation (JSON), is uploaded to the original d.store platform.

The platform uses Semantic Web technologies to represent concepts of collaboration artifacts, such as emails or wiki pages, as ontologies. The concepts are linked through associations, e.g. a person is linked to an email by being its sender. They are also time-annotated and, accordingly, can be put in correlation with project timelines. Ontologies define which kind of data can be uploaded to the platform. Gathered artifacts are combined into so-called team collaboration networks (TCN) [18], which provide a unified representation of the digital collaboration happening within project teams. The TCN are persisted within an RDF graph database[1].

3.3 Data Analysis

The unified network structures allow users of the platform to incorporate all captured means of digital collaboration into the analysis and iteratively explore the data. For example, this structure allows advancing from question like "How many emails have been sent during a certain period of time" to more complex ones, such as "How many emails have been sent that referenced a wiki page, which has been edited frequently by more than x collaborators", without any adoptions of the stored data. Data querying is possible through a SPARQL [11] interface.

We are, furthermore, exploring extended analysis capabilities, such as such as natural language queries based on the available ontologies [5] or graphical data representations, which could lead to the detection of collaboration behavior that needs further investigation through visual perception of anomalies in the created diagrams.

3.4 Sharing of Analysis Results

As stated earlier, core functionality of the proposed system is the creation of a database of collaboration traces. This implies that all collaboration data uploaded to the platform is stored permanently along with additional information about the projects that it was recorded at. If other researchers want to verify findings they made within their own datasets, they can use this database to select a suitable sample set of projects and run the corresponding queries against all stored team collaboration networks.

In order to be able to select suitable sample projects for the questions at hand, data about the projects needs to be available. Problem domain, team size, team structure, project duration, or success metrics are just some examples for possible parameters. This data has to be entered by platform users upon creation of new projects.

The stored collaboration data is of sensitive nature as it allows to make assumptions about the working behavior of individuals. However, without further context those assumptions might not properly reflect project reality and lead to wrong conclusions. Therefore, only the original uploader of certain data can see the original names, addresses, and contents, while all other people accessing the collaboration events will only be able to see obfuscated data. In addition to this on-the-fly obfuscation, the platform also allows to pre-obfuscate the data before storing it within the database. Furthermore, it is possible to refuse storage of data in the local database of the platform. Instead, the URI of a compatible data store (i.e., one that exposes a SPARQL interface) has to be specified by the platform user.

Sharing of collected collaboration data and project information is just one aspect of the shared platform approach. The other one is sharing of analysis results. Therefore, we introduce a notion of collaboration patterns based on [10]. Within the TCN, collaboration traces are stored. They represent concrete sequences of collaboration events. Collaboration patterns, on the other hand, are abstractions of traces. For example, "A remote team member sends an email that contains a link to a newly created wiki page in a timeframe of 60min after page creation" is a generalization of "Bob

[1] AllegroGraph RDFStore, `http://www.franz.com`

from the US-based team sends an email containing a link to the wiki overview page, which he created at 3:30, to the team email list at 3:45". The formal definition of this notion of collaboration patterns is currently in progress and will be submitted for publication, shortly.

If researchers or other users of the platform identify collaboration patterns that have proven to be beneficial or detrimental for project success, they are able to store them in a central pattern repository provided by the platform. Other users are, in turn, able to access those patterns and analyze their own TCN for possible occurrences.

4. Platform Applications

The system is currently applied within two case studies. Both studies test different aspects of the platform and help us in identifying possible extension points for further development. The case studies differ in both problem domain and project setups, and, hence, also showcase the versatility of the platform.

4.1 Software Engineering

An early version of the platform has already been applied in the course of a software engineering exercise that resembled software development in a mid-sized development team [7]. During this application, it became apparent that the digital collaboration activities of the students provide subtle insights into their working behavior that were difficult to obtain through manual observation processes, such as interviews or questionnaires.

Building on this experiences, the extended platform is currently applied within a new iteration of the lecture. Contrary to the first installment, the exercise features two development teams that compete with each other in developing a customer relationship management system. Both teams consist of approximately 50 students, are further subdivided into eight sub-teams, use Scrum [13] as the development process, and are equipped with similar groupware tools.

By that, comparability of the respective development teams is greatly enhanced, and the the case study provides us with the possibility to detect different collaboration patterns shown by the two development teams. Again, an intensive manual observation process is performed to be able to relate the collected collaboration activity records to real life events not captured by the system (e.g., special events at the institute or contents of lectures that accompanied the exercise).

4.2 Engineering Design

A second case study is performed in the course of an engineering design class [2]. In this course, teams of six to eight students from two different universities work on a design task given by a corporate partner. The teams are usually separated over different continents and have to deal with the challenge of working distributed over different time zones. Over the course of nine months the teams have to organize their work, develop ideas and, finally, build a working prototype of their idea. The course follows Design Thinking methodology and is split up into roughly three different phases, which require varying communication and collaboration techniques. While all teams work on different design tasks, the course framework provides a certain degree of comparability of the projects.

This application provides the foundation for the formalization of collaboration patterns. Due to the project setting, ad-hoc, informal collaboration is very rare, and the majority of collaboration activity is actually performed using observable digital media. Hence, the different working behavior that the teams develop over the course of the project is clearly visible within the collaboration traces and can be compared to the working style displayed by other teams of the same year or previous iterations of the course.

5. Conclusion and Future Work

In this paper, we have presented a platform that supports researchers in the field of virtual collaboration research, as it gives them access to collaboration data from a variety of projects without requiring them to perform the underlying case studies or experiments by themselves. By sharing collaboration traces of analyzed projects, a database is created that allows for sample sizes, which could not be achieved by individual efforts.

The simple workflow that is required to upload collaboration traces and analyze them by means of a standardized query language fosters platform usage even by researchers without technical experience.

Furthermore, the platform not only allows to share sets of collaboration data with other researchers, but enables publication of analysis results by means of collaboration patterns. Collaboration behavior that might has positive or negative effects on project execution and possibly even project success can be stored in a generic manner and newly uploaded projects can be analyzed for the occurrence of such patterns.

The system is being tested in sample applications in the fields of software engineering and engineering design. Those case studies help us to identify possible points for future extensions, as well as areas that require further research. In addition to the extended analysis functionality mentioned in Section 3.3, this includes the application of the platform during the runtime of engineering projects. It has to be determined if automatic matching of collaboration patterns and, thus, detection of detrimental or beneficial collaboration behavior in early project stages render the platform a viable tool for the management of, for example, large, distributed software engineering teams.

References

[1] Syed SR Abidi, Ashraf Abusharek, Ali Daniyal, Mei Kuan, Farrukh Mehdi, Samina Abidi, Faisal Abbas, Philip Yeo, Farhan Jamal, and Reza Fathzadeh. A service oriented e-research platform for ocean knowledge management. In *Proceedings of the 2010 6th World Congress on Services*, SERVICES '10, pages 32–39, Washington, DC, USA, 2010. IEEE Computer Society.

[2] T. Carleton and L. Leifer. Stanford's me310 course as an evolution of engineering design. 2009.

[3] Kadir Alpaslan Demir. *Measurement of software project management effectiveness*. PhD thesis, Naval Postgraduate School, Monterey, Calif., 2008.

[4] John R. Dixon. On research methodology towards a scientific theory of engineering design. *AI EDAM*, 1(03):145–157, 1987.

[5] David W. Embley and Roy E. Kimbrell. A scheme-driven natural language query translator. In *Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, CSC '85, pages 292–297, New York, NY, USA, 1985. ACM.

[6] Kevin Forsberg, Hal Mooz, and Howard Cotterman. *Visualizing project management : a model for business and technical success*. Wiley, New York, 2000.

[7] Thomas Kowark, Jürgen Müller, Stephan Müller, and Alexander Zeier. An educational testbed for the computational analysis of collaboration in early stages of software development processes. In *Proceedings of the 44th Hawaii International Conference on System Sciences (HICSS)*, January 2011.

[8] Microsoft. http://msdn.microsoft.com/en-us/vstudio/ff637362, Visual Studio Team Foundation Server 2010, 2011.

[9] Masao Ohira, Reishi Yokomori, Makoto Sakai, Ken-ichi Matsumoto, Katsuro Inoue, Michael Barker, and Koji Torii. Empirical project monitor: A system for managing software development projects in real time. In *International Symposium on Empirical Software Engineering*, Redondo Beach, USA, 2004.

[10] Nikos Papageorgiou, Giannis Verginadis, Dimitris Apostolou, and Gregoris Mentzas. A collaboration pattern model for virtual organisations. In *PRO-VE*, pages 61–68, 2009.

[11] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF, W3C Recommendation, http://www.w3.org/TR/rdf-sparql-query/.

[12] Kurt A. Reiner. *A framework for knowledge capture and a study of development metrics in collaborative engineering design*. PhD thesis, Stanford, CA, USA, 2006.

[13] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.

[14] Ching seh Wu, Wei chun Chang, and Ishwar K. Sethi. A metric-based multi-agent system for software project management. *Computer and Information Science, ACIS International Conference on*, 0:3–8, 2009.

[15] Philipp Skogstad. *A Unified Innovation Process Model for Engineering Designers and Managers*. PhD thesis, Stanford University, Center for Design Research, 2009.

[16] Matthias Uflacker. *Monitoring Virtual Team Collaboration: Methods, Applications, and Experiences in Engineering Design*. PhD thesis, Hasso Plattner Institute for IT Systems Engineering, Potsdam, Germany, 2010.

[17] Matthias Uflacker, Thomas Kowark, and Alexander Zeier. An Instrument for Real-Time Design Interaction Capture and Analysis. In C. Meinel and L. Leifer, editors, *Design Thinking: Understand – Improve – Apply*. Springer (in print), 2010.

[18] Matthias Uflacker and Alexander Zeier. A semantic network approach to analyzing virtual team interactions in the early stages of conceptual design. *Future Generation Computer Systems*, 27(1):88–99, 2010.

Refactoring Catalog for Legacy software using C and Aspect Oriented Language

S.A.M Rizvi and Zeba Khanam,
Jamia Millia Islamia,
New Delhi, India

Abstract- *This paper explores the combination of AOP and refactoring, two techniques that deals with the problems of permanent evolution of software. AOP has evolved to deal with a large legacy of object-oriented (OO) code. Most of the work has been done in the area of refactoring the object oriented code but little with the procedural code and that too with aspect orientation. But the AO concept when applied to the procedural source code can be used to solve the existence of potential problems in the code thus improving the underlying design. The poorly designed procedural code when refactored with aspect orientation yields a better code. This paper documents a collection of novel refactorings enabling the extraction of crosscutting concerns from the procedural legacy source code. Therefore we establish a systematic approach for refactoring and then propose a refactoring catalog to deal with the procedural code that establishes a systematic approach to refactor the procedural code. To this end, we first investigate the impact of existing object-oriented refactorings such as those proposed by Fowler on procedural code. Then we propose some novel refactorings that are unique to the procedural code and the ones using the aspect concept are termed as aspect-oriented refactorings. We use AspeCt oriented C, an AOP language for C to demonstrate our approach.*

Keywords: Refactoring, legacy software, aspect oriented programming, object oriented refactoring

1 Introduction

Refactoring is an approach that facilitates the continuous change of source code, enabling it to evolve in line with changes in environments and requirements. [1]

Software engineering is a discipline that is dedicated to designing, implementing and modifying software so that it is of higher quality, more affordable, maintainable and easier to build. It creates models and tools for the support of abstraction [12][13]. We cannot concentrate on many subjects or concerns simultaneously. Instead, we need to abstract from most of the concerns so that we can concentrate on a particular concern in order to deal with it effectively. The abstraction can be achieved by applying the separation of concern technique. Loose coupling is an essential property of any software. We need to break, or decompose systems and problems into smaller subsystems and sub-problems, in order to concentrate on each sub problem by turns. The more it is loosely coupled; the easier it is to break. Loose coupling greatly facilitates system development because we are able to reason with an issue of the system at a time. AOP [1][14] provides stronger modularization. Experience with refactoring software [4] [11][7] suggests that refactoring techniques [5] have the potential to bring the concepts and mechanisms of aspect orientation to existing procedural code and applications, hence implementing the "separation of concern" in a more effective way.

However, although refactoring has been studied widely for object-oriented software. The development of refactoring patterns for procedural software using the concept of aspect oriented programming has not been done. In this paper we, propose a systematic approach to refactoring procedural code and a refactoring catalog for procedural software.

2. Activities of Refactoring Process

AOP is an emerging field and its steady progress can be traced from "bleeding edge" research field to mainstream technology [1][24]. Its contribution to refactoring is remarkable and especially its support to refactoring object oriented frameworks provided stronger modularisation and software composition mechanisms than those provided by previous technologies. But one of the hurdles in its implementation to the procedural code is that there isn't any developed or established theory and discipline for the process and that it does not have that much of modularity as it is in object oriented code due to the presence of classes.

In this paper we propose a systematic approach towards procedural programs. The refactoring approach is based on Monteiro's pattern language. In [25] the pattern language contributes to refactoring existing systems into aspect

oriented versions of those Systems. The pattern language consist of three patterns Cross cutting concerns, Decide to refactor to aspects, and Refactor towards Aspect friendly code base that are intended to focus on the initial issues that arise when considering the option to refactor an existing OO system to AOP.

When performing refactoring on the procedural programs there are certain issues that need to be assessed before refactoring.

1. Is the refactoring process focusing only on refactoring cross cutting concern (CCC)?

2. What about the code written in bad style or poorly designed code, should it be refactored or not?

3. Should the badly designed code be refactored into aspect friendly code or should it be refactored according to the traditional approach?

Refactoring takes time but can be performed in phases. Refactoring to a better style or design brings benefits to understandability, maintainability and ease of evolution, that are independent of whether you Decide to refactor to aspects or not.

Before going ahead with AOP refactorings, ensure that the system is already well-decomposed according to the current notions of good style. Fowler's book provides a collection of 22 bad smells that indicate the kinds of situation in the code that warrant the use of the refactorings. Good examples of such smells are: Duplicated Code, Long Method and Large class. But these refactorings are for object oriented code

In the subsequent section, refactorings for procedural programs are proposed. The problem is studied from three aspects.First; it is investigated [14] whether the object-oriented refactorings such as those proposed by Fowler [4] can be applied to procedural code. Second; proposal of some modification guidelines in order to adopt some of these refactorings to the domain of procedural language. Final step is the identification of some new refactorings using ACC that are unique to procedural C code.

3. Refactoring procedural code

The motivation for using AOP for refactoring is that it is a new language paradigm that is different from procedural and object-oriented language; it provides a new construct, an aspect, to modularize crosscutting concerns in the code.

Legacy code is a critical part of many systems and the source of many problems. Software systems must constantly adapt to changes of the environment in which they operate. In most cases, legacy software has been developed in obsolete languages by using old-fashioned development practices. This complicates the upgrade of legacy parts. The most popular way to apply AOP in software development is to extend existing programming languages with new constructs, the most important of which are pointcuts, advices and introductions, that make it possible to express the crosscutting functionalities. There are aspect versions of many widely used languages as, for instance, Java, C/C++, Ada, Smalltalk, Perl, Python, etc.Opdyke identified a series of refactorings for object-oriented source code, using C++ as the subject language. But a similar catalog for procedural languages like C does not exist. So we explore a catalog of refactorings for procedural programming using C and AspeCt Oriented C.The table below depicts the impact of applying the object oriented refactorings on procedural code. Some of them are applicable directly wheras some require modification when applied using the aspect orientated concept. The table 1. below illustrates some of them:

Table 1: Impact of applying object oriented refactoring on procedural code

Extract method (no change)	Encapsulate Field (change)
Add a method (no change)	Move Field (change)
Delete unreferenced method (no change)	Shotgun surgery (change)
Add Parameter to method (change)	Consolidate conditional expression (change)
Rename method (change)	Merge duplicate conditional code (change)
Inline method (no change)	Extract Conditional (change)
Hide Method (change)	Replace Iteration with Recursion (no change)
Move Method (change)	Replace Error code with Exception (change)
Introducing Divergent Change in methods(change)	Replace type code with strategy(change)
Remove method (change)	Simplify conditional Expression (change)

3.1Object Oriented Refactorings for procedural code

We study the impact of object oriented refactorings [4] (those applicable) on the procedural programs and observe the changes that are required to implement them on the code. Some of those are directly applicable without any change whereas some are changed when aspect oriented concepts are applied.

Motivating Examples

We present here a few examples to explain the behavior when applying existing object-oriented refactoring (OO-refactoring for short) to procedural code and the changes that we have incorporated to refactor the code.

When things go wrong the program can be stopped with an error code. Fowler refactored with exception handling techniques .By carefully examining the refactoring *Replace Error code with Exception* [4], we found that this could not be achieved in procedural languages like C that did not support the exception handling mechanism. As a result, users of these legacy programming languages often implement exception handling by applying an idiom. An idiomatic style [22] of implementation has a number of drawbacks: applying idioms can be fault prone and requires significant effort. It is elaborated in the example below how we have changed this refactoring and applied it to the procedural code as well. Figure 1 presents here function named *withdraw(int amount)* that performs a check everytime an amount has to be withdrawn and returns -1 everytime the amount exceeds the balance. In C a special output is used to indicate error .The developers traditionally use a return code to signal success or failure of a routine. But with ACC, we examined that we can omit the return code idiom with try () and catch() advice. This makes programs easier to understand, thus reducing the complexity of the code too.

Before refactoring

```
int withdraw (int amount)
{
if (amount > _balance)
        return -1;
else        {
balance -= amount;
 return 0;}
```

After refactoring with aspect code

```
void withdraw (int amount) {
balance - = amount ;}

before ():call (void withdraw (int)){
if(amount>balance)
throw ();}

catch ():try (call (void withdraw(int)))
   {printf ("Invalid amount");}
```

Figure 1: Replace Error code with Exception

The next example that we present is extraction of conditional statement into advice. This refactoring has been modified in order to enable condition checking dynamically and free the method containing the conditional statement from unnecessary complexity. We depicted the case where it could be applied. A conditional statement controls the execution of the marked code. Fig. 2 shows this refactoring. The conditional statement in Fig. 2 in method g() determines the execution of another function f() in the control flow of g().This conditional statement can be made a part of advice and checked every time the function f() has to be executed.

Before Refactoring

```
void g()
{
-
-
if (value==TRUE)
{
f();}
```

After Refactoring

```
void g()
{
-
-}
pointcut checkvalue():call f() && infunc g();

before: p(){
if(value==TRUE)
f();}
```

Figure 2: Extract conditional

A Catalog of aspect oriented refactorings code

The most obvious set of aspect-specific refactorings are simply straightforward analogs of refactorings that apply to classes or members. But its not true in case of procedural code whose structure is completely different from object oriented code and is completely dependant on methods and procedures. Aspect-orientation [23] introduces new aspect-aware refactorings that differ from existing object-oriented refactorings.

Table 2 lists aspect-oriented refactorings we proposed; This is not the complete set of aspect specific refactorings that we identified, more AO- refactorings will be added to the list as we go on with our experiments and get some new result.

Table2: A catalog of aspect oriented refactorings for procedural code

Extract fragment into advice	Create introduce() advice
Create Empty Advice	Add an intype( ) poincut
Extract Common Members to Aspect	Create an around advice
Create Named Pointcut	Move Local Member Declaration to Aspect code
Rename a Named Pointcut	Refactoring idiomatic code to AO specific style with a try() Pointcut
Introduce a primitive pointcut	Introduce catch () Advice
Combine pointcut	Create an empty advice
Introduce a Preturn () in advice.	Replace method with advice
Create a cflow pointcut	Replace type code with advice
Create multiple advice for different situations	Extract method into advice

Motivating Examples

In this section 3 examples are presented to demonstrate refactorings with AOP.We have used AspeCt oriented C constructs for the purpose. An example (fig 3) demonstrates the extraction of an advice from the code. Extract fragment into advice case is discussed in the next section for elaborating the refactoring.

Figure 3 contains a code snippet that demonstrates the extraction of a method in to advice. The figure depicts a push operation on a stack that calls the display () method after push operation that is called at many other places. In order to reuse the condition, we may turn it into a piece of after advice whose name reflects the value that the advice gives. The recommended action in this case is to create a pointcut capturing the required joinpoint and context and move the code fragment to an appropriate advice based on the pointcut.This refactoring should be used when we want to move to an aspect a piece of functionality that does not comprise a complete method or sometimes it is a simple method call. In some cases of object oriented code, the *Extract Method* [4], Fowler) can be conveniently applied. However, even in such occasions there will be need to create an advice that calls the method. However before copying the code fragment, a careful analysis of the method's (or, sometimes, the constructor's) body should be made, in order to find a suitable pointcut to capture the exact set of intended joinpoints. If the primary code does not offer a suitable joinpoint, one or more refactorings must be performed until the code is ripe for this refactoring. Sometimes the advice will need to capture local variables (either primitives or object references). Note that these situations may be a sign that the method is more complicated than it should be. Consider whether it would make sense to split it in various parts, using *Extract Method* [4] for each part in turn. Such a split may provide the joinpoints you need.

.

Before Refactoring

```
public void push (int data)
{
array [++_top] = data;
display();}
```

After Refactoring

```
public void push(int data)
{
array [++_top] = data;
display ();          }

pointcut callpush (int data): execution (void push( int)) &&
args(data);

after(int p )  : callpush(p)
{display ();}
```

Fig 3: Refactoring for extraction method into advice

Figure 4 presents another program that depicts the extraction of a named pointcut.This is done in 2 steps [10] The first step, uses refactoring to create the initial pointcut and advice instances necessary to aspectize the code selected. The second step combines these instances into the final pointcuts and advices. The basic aim of creating a named pointcut is to make the pointcut more readable, usable and easier to be reused at different places. The example depicts the extraction of pointcut.The beginning of the method has been extracted. There are variants to this also like end of the method handler ,extract before/after call that we have modified and implemented in [20].Figure 4 contains a method student which has a precondition declared by if statement. A conditional statement controls the execution of the marked code. The conditional statement if (id= = null) is considered to be part of the advice, as it

determines the execution of the method (adddetails (details)) every time it is invoked. In order to reuse the precondition, we may turn it into a piece of before advice whose name reflects the value that the advice gives. This can be refactored by extracting a pointcut check to represent the join point related to the advice. Thus, it becomes a checked condition incorporated into an advice (using the ACC syntax).By doing so; the extracted pointcut check can be reused by other advice as well. We call such a refactoring as Extract Pointcut.

Before refactoring

```
void student (int id) {
if (id= = null)
{
printf (``The argument is null");
break();
}
adddetail(details);
}
```

After refactoring

```
void student (int id) {
adddetail (details);}

pointcut check (int id): call (void student (int id));
before (): check (int id){
if (id = = null)
{ printf("The argument is null");}}
```

Figure 4: An AO-refactoring for extraction of a named pointcut.

Another example presented in figure 5 depicts *extract fragment into advice*. In C it is common practice to check the return value after memory allocation to ensure the return value is non null. This code often looks as in figure 6 before refactoring which is taken from [9]. This condion is specified after each call to malloc ().Furthermore, similar checks apply for other memory allocation calls, such as calloc () and realloc (). Since malloc () is a function that is used very frequently for dynamic memory allocation, the above code fragment is scattered across the whole system. In order to reuse the condition, we may turn it into a piece of after advice whose name reflects the value that the advice gives. So this check will be made everytime the malloc() or calloc() functions are called.

Before Refactoring

```
int *x ;
x = (int *) malloc (sizeof (int) * 4);
<--- dynamic memory allocation
if (x = =NULL)
{
/* rountine to handle the case when memory allocation
failed */
}
 /* routine for handling the normal case */
```

After refactoring

```
after (void * s): (call ($ malloc (...)) || call ($ calloc (...)) ||
call ($ realloc (...))) && result(s)
 {              char * result = (char *) (s);
                if (result == NULL) {
/* routine to handle the case when memory allocation fails
*/
                }}
Now, the core program looks as follows:
...
int *x;
x = (int *) malloc (sizeof (int) * 4);
                                <--- dynamic memory allocation
/* routine for handling the normal case */
```

Figure: 5- Extract fragment into advice

4. Related Work

In this section the related work in the area of refactoring for object-oriented, aspect oriented and procedural programs that bear relation to our work is highlighted.

Refactoring object oriented programs [6][8] is an active research area. The major contributions being done by Fowler. Then refactorings were done with AO concept based on the same structure as [7]. Van Deursen et al. [18] provide an overview of the state of art in the area of aspect mining and refactoring. Monteiro identified how can existing code smells be used to identify candidate refactorings and how can the introduction of aspects be described in terms of a catalogue of new refactorings.M.Cecato has also worked on migrating the object oriented code to aspect oriented code. In that the concept of aspectizable interfaces was explored. Binkley et al in their work [17] devised a human-guided approach to refactoring object-oriented programs to the aspect-oriented paradigm.

Then refactoring of aspect oriented programs has also become a major research area with several groups studying it with a different approaches and viewpoint. Wloka [15] explored the relationship between refactoring and aspect-orientation, but did not directly address the issue on how to perform refactorings on aspect-oriented programs. Borba and Soares [3] proposed a program transformation based approach to developing refactoring and code generation

tools for Aspect. Iwamoto and Zhao announced in [16] their intention to build a catalogue of AO refactorings.

Wide research has also been done on refactoring of procedural code but no work is done till now to refactor procedural legacy code utilizing the aspect oriented concept. Abadi and Feldman [2] demonstrated the utility of the mutual refactoring of procedural code (using C code) and statecharts in two case studies.Marchetti etal has worked on refactoring legacy systems using component based architecture [20].The contribution by Mortenson [21] was also based on refactoring legacy code but the issues addressed were to evaluate the effects of aspects on program maintainability and cost factors of adopting AOP, including performance, testability, and defect introduction. Most of the work has been carried on object oriented legacy application very scarce amount of work is available on refactoring the procedural code.

Thus the refactoring catalog is first of its kind. In this paper only few refactorings are elaborated but we are already in the process of elaborating it.

5 Conclusion

In this paper we proposed a systemic approach to refactor legacy software with procedural code like C. To this end, we first investigated the impact of existing object-oriented refactorings such as those proposed by Fowler on procedural programs and gave some guidelines for solving these problems. We then proposed some new aspect-oriented refactorings that are unique to procedural code but different from existing object-oriented refactorings. The goal of this code change is to remove the drawbacks due to the presence of crosscutting concerns. Traditional languages fail in modularizing functionalities that are tangled and scattered with respect to the principal decomposition offered by the language in use. Especially with the procedural languages like C, refactoring is a more complex job they are not even equipped with the object oriented features of classes and objects.However, although refactoring has been studied widely for object-oriented software, The development of refactoring patterns for procedural software that also with aspect oriented techniques has not been done .Though several researchers [3, 5, 9, 12,24] are working on aspect oriented and object oriented refactoring , but no work has yet been done on refactoring of procedural programs .

In this Ph.D. project we research refactoring techniques for procedural code and have proposed a refactoring catalog using C and ACC. The refactorings are AO specific. We aim to build a collection of refactorings and code smells capable of expressing the characterisation of a new programming style (aspect specific), in the same way the refactorings and code smells documented by Fowler et al. successfully represent a notion of good style for OO source code.We have done a review of the traditional OO and procedural code smells in light of AOP. The proposal of several novel code smells, including the ones that are specific to aspects. We are not considering mechanisms for automatic support, but rather aiming to pinpoint and characterize the operations as performed manually. Pertinent issues include finding the most useful transformations, their mechanics, which preconditions must be met prior to each refactoring, and how the structure of the legacy code may influence choice of the next refactoring to apply .We are already in the process of sculpturing the refactoring catalog, which we expect to complete positively before writing our doctoral dissertation.

6 References

[1] Elrad T. (moderator) with panelists Aksit M., Kiczales G., Lieberherr K., Ossher H.,Discussing Aspects of AOP. Communications of the ACM, pp. 33-38, October 2001.

[2] Moria Abadi and Yishai A. Feldman., Refactoring in Multiple Representations: Code and Statecharts. 2009

[3] P. Borba and S. Soares. Refactoring and Code Generation Tools for AspectJ. October 2002.

[4] M. Fowler. Refactoring: Improving the Design of Existing Code. Addison-Wesley Longman, 1999

[5] Anders Hessellund .Refactoring as a Technique for the Reengineering of Legacy Systems, 2004.

[6] E. Kodhai1, R. Ragha Sudha, K. Reka and A. Amudha. AOP based Refactoring of Java Legacy System. Proceedings of the International Joint Journal Conference on Engineering and Technology (IJJCET 2010)

[7] Opdyke W. F., "Refactoring Object-Oriented Frameworks, Ph.D. Thesis, University of Illinois at Urbana-Champaign, USA, 1992

[8] Miguel Monteiro, "Refactorings to Evolve Object-Oriented Systems with Aspect-Oriented Concepts", AOSD, 2004.

[9] www.aspectc.net

[10] M.Ceccato. Migrating Object Oriented code to Aspect Oriented Programming, IEEE International Conference on Software Maintenance, ICSM 2007.

[11] Griswold W. G., Program restructuring as an aid to software maintenance. PhD thesis, University of Washington, USA, 1991.

[12] Parnas D. L., "On the criteria to be used in decomposing systems into modules. Communications of the ACM, pp. 1053-1059, December 1972.

[13] Dijsktra E., A Discipline of Programming, Prentice Hall, 1976.

[14] Rizvi S.A.M, Khan M.U and Khanam Z. "Refactoring using Aspect orientation for Object oriented code: A comparison with object oriented refactorings".

Proceedings of IEEE 2010 International Conference on Computer and Computational Intelligence, ISBN: 978-1-4244-8948-0, IEEE Xplore, 2010.

[15] J. Wloka. Refactoring in the Presence of Aspects. ECOOP2003 PhD workshop, July 2003

[16] M. Iwamoto and J. Zhao. A Systemic Approach to Refactoring Aspect-oriented Programs. August 2003.

[17] D. Binkley, M. Ceccato, M. Harman, F. Ricca, and P. Tonella, "Tool-supported refactoring of existing object-oriented code into aspects," *IEEE Transactions on Software Engineering*, vol. 32, no. 9, pp. 698–717, 2006.

[18] A. Van Deursen, M.Marin, and L. Moonen. Aspect Mining and Refactoring. In *Proceedings First International Workshop on REFactoring: Achievements, Challenges, Effects*. University of Waterloo, 2003

[19]M.Marin, L.Moonen, and A. van Deursen. An Integrated Crosscutting Concern Migration Strategy. and its Application to JHotDraw. In *Proceedings Seventh International Working Conference on Source Code Analysis and Manipulation*, pages 101-110, IEEE Computer Society, 2007

[20] E.Marchetti, F.Martelli and A.Polini, "Refactoring a legacy system using components", 2003.

[21] Michael Mortensen, Sudipto Ghosh, "Aspect-Oriented Refactoring of Legacy Applications: An Evaluation", 2010

[22] M. Bruntink, A. van Deursen, and T. Tourwe, "Isolating idiomatic crosscutting concerns," in ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance. Washington, DC, USA: IEEE Computer Society, 2005, pp. 37–46.

[23] Rizvi S.A.M and Khanam Z., Introduction of AOP techniques for refactoring legacy software , International Journal of Computer Application, 1(7):90-94, DOI, Published by Foundation of Computer Science, United States, ISSN: 0975-8887,25th Feb, 2010

[24] Sabbah D., Aspects – from Promise to Reality. Proceedings of the 3rd International Conference on Aspect-Oriented Software Development (AOSD 2004), Lancaster, UK, ACM press, pp. 1-2, March 2004

[25] Miguel P. Monteiro, Ademar Aguiar "Patterns for Refactoring to Aspects: An Incipient Pattern Language, Proceedings of the 14th Conference on Pattern Languages of Programs (2007), pp. 1-10. doi:10.1145/1772070.1772079, 2007

SESSION

SOFTWARE ARCHITECTURE AND DESIGN PATTERNS

Chair(s)

TBA

Design Patterns – A Modeling Challenge

Vojislav D. Radonjic, Soheila Bashardoust, Jean-Pierre Corriveau, and Dave Arnold
School of Computer Science, Carleton University
Ottawa, Canada
radonjic@acm.org

Abstract – *Design patterns and their catalogs are an important phenomenon in software development, and they present a challenge to modeling techniques and supporting tools. In this paper we identify and illustrate the problem in terms of the creational family and one of its members, the Factory Method.*

Keywords: *design patterns; variability modeling; variant selection*

1. Introduction: Design Patterns

Designers are bridge builders from the problem world of requirements to the solution world of various candidate systems. Design patterns have emerged as important tools for documenting and sharing the 'bridge building' experience of spanning from the side of requirements (scenarios and features) to the side of implementations (C++, Java, various run-times). Like craftsmen in other disciplines, software designers place great importance on their tools, and judging by the large number of copies of the GoF catalog [3] sold, and by the ubiquity of these patterns in newly developed systems, these craftsmen have chosen design patterns as one of their essential tools.

Design patterns, however, remain a challenge for the modeling community. The core pattern conceptualization – the often repeated 'a design solution to a problem in context' – is considerably more complex, as we will see in Section 2. In Section 3 we summarize the modeling challenges.

2. Design Patterns: a Challenge to Modeling

Design Patterns are complex abstractions that span from requirements through design to implementation, with variability at each phase. In the remainder of this section we illustrate the challenge of modeling the creational family, and in particular, the Factory Method from the GoF [3]; our modeling goal is to explicit the concepts, their variations and connections as they appear in various sections of the template format used in the GoF catalog.

2.1 Example: Design Level Requirements For The Creational Family Of Patterns

Let us start at the source, the GoF catalog[3].

"Creational design patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented. ...

Creational patterns become important as systems evolve to depend more on object composition than class inheritance. As that happens, emphasis shifts away from hard-coding a fixed set of behaviors toward defining a smaller set of fundamental behaviors that can be composed into any number of more complex ones. Thus creating objects with particular behaviors requires more than simply instantiating a class." p81 [3]

The above two paragraphs begin the presentation of creational patterns: they establish the fundamental **goal** of this family of patterns as ***System Creation*** in a *specific context*. The context is what leads to a particular kind of refinement of the goal of system creation that is based on object-orientation; where a system is composed of objects (components), whose structure and behavior are defined in terms of classes (types). It is this context that governs the space of design mechanisms used to achieve the goal.

"There are two recurring themes in these patterns. First, they all encapsulate knowledge about which concrete classes the system uses. Second, they hide how instances of these classes are created and put together. All the system at large knows about the objects is their interfaces as defined by abstract classes. [component types] Consequently, the creational patterns give you a lot of flexibility in what gets created, who creates it, how it gets created, and when. They let you configure a system with "product" objects that vary widely in structure and functionality. Configuration can be static (that is, specified at compile-time) or dynamic (at run-time)." p81 [3]

The goal of *System Creation* is that of creation and initialization of the system structure out of a set of component types. The previous paragraph mixes the what and the how, the goal and the mechanisms used to achieve it. Our aim is to separately model the goal of system creation from the mechanism used to achieve it.

From a designer's perspective, the goal of system creation is refined in terms of component types and their relationships (family constraints, initialization ordering constraints, incompatibilities between component types, etc.), system object structure, binding place and time. The forces leading to selection of one pattern over another, therefore, are in these terms:

- Sets of component types: static, dynamic, family of component types
- System Structure
- Uniqueness (Singletoness)

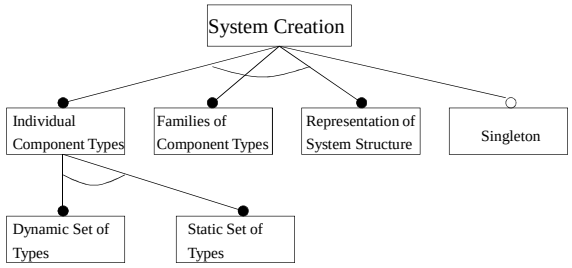


Figure 1: Top-level Feature Model for the Creational Family

Fig. 1 models system creation in terms of a feature model, commonly used in domain engineering [7]. It starts at the root with "System Creation" and refines to a tree of concepts that embodies all the creational patterns.

In the context of design patterns, domain analysis is about building a space of forces that is a starting point for selecting a design variant. The feature model in Fig. 1 models that space of forces. The syntax and general notational semantics are not important — rather, what is important is the internal relations within the feature model and, in particular, its overall relationship to the design space in which the design variants of a pattern live. *Most of the content found in the patterns of GoF is in that relationship of forces to variants.* The problem in the GoF catalogue is that that relationship is not clearly expressed. The aim of the coming description is to explicit the relationship of forces to variants. With that in mind, it is critical to keep the point of view of the designer as the primary customer of this modeling exercise.

The notion of a factory object is not represented in Fig. 1, because factories are the "how" part of the pattern, which we are explicitly leaving out of Fig. 1. Put another way, a factory is a solution abstraction, not a problem abstraction. Factories are a generalization of creation of an object to 1) creation of any object that fulfills a particular interface (i.e. type), 2) creation of a system of objects (abstract factory, builder, ...).

In Fig. 1 the terms *component* and *component type* are used as opposed to objects and classes to indicate independence from OO mechanisms. Specifically, should the following content be part of the feature model?

"Creational patterns become important as systems evolve to depend more on object composition than class inheritance. As that happens, emphasis shifts away from hard-coding a fixed set of behaviors toward defining a smaller set of fundamental behaviors that can be composed into any number of more complex ones. Thus creating objects with particular behaviors requires more than simply instantiating a class. " p81 [3]

The above paragraph defines the context of system creation in OO rather than the design mechanisms used in the patterns, and, therefore, its content is eligible for feature modeling. On the other hand, the following paragraph from the concluding discussion about creational patterns is filled with solution mechanics of each pattern, which should not make it into the feature model directly:

"There are two common ways to parameterize a system by the classes of objects it creates. ….

Prototype has the factory object building a product by copying a prototype object. In this case, the factory object and the prototype are the same object, because the prototype is responsible for returning the product."p135 [3]

But the following closing paragraph of the creational patterns would make it into feature modeling because it addresses design-level requirements directly:

"Designs that use Abstract Factory, Prototype, or Builder are even more flexible than those that use Factory Method, but they're also more complex. Often, designs start out using Factory Method and evolve toward the other creational patterns as the designer discovers where more flexibility is needed. Knowing many design patterns gives you more choices when trading off one design criterion against another." p136 [3]

The above paragraph is particularly strong in saying how to navigate and select—being able to easily navigate a catalogue is important to allow the designer to build and refresh a mental map of the design space. This is not simply a surface-level usability issue, although it may exhibit itself as such, but a much deeper semantic one. As the scale of design-spaces grows (witness the scale of options available in J2EE, .NET, etc.), the decision-making support offered by usable cognitive models becomes more than just a deep academic exercise, it becomes a critical practical issue.

It may appear that the only sources for the Fig. 1 Feature Model would be the Intent, Motivation and Applicability sections of the GoF format [3] but, surprisingly, the one that yields the most information is the Implementation section.

2.2 Example: Design Level Forces For The

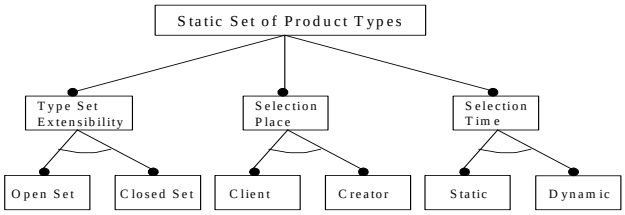


Figure 2: Factory Method Feature Model

Factory Method Pattern

The Factory Method Feature Model (Fig. 2) is the result of domain analysis of the pattern, and fits within Fig 1. (Static Set of Types box). The analysis leading to Fig. 2 focuses on the problem space of a design pattern, and has as its primary customer the designer in a role of a navigator towards a potential solution. Consequently the resulting feature model is a kind of selection/navigation mechanism. There is, however, a deeper aspect to feature modeling — a representation of context within which the solution space of a pattern exists. This context is multi-faceted and dependent

on perspective of the user of the pattern: extensibility, performance, resource costs, etc.

Consider that a designer arrives to this pattern under forces that push towards the overall creational pattern space. The selection of one pattern over another, and one variant over another takes place as a result of differentiation and specialization of the forces: the overall goal of creating a system of components is differentiated into forces described in Fig. 1.

The notion of product type set is central to the Factory Method pattern as well as Abstract Factory and Prototype. [3] The objects that are created (often to serve as delegates to some client) have types from that set of product types. In Factory Method pattern the set is statically bound, i.e. the collection of types is determined before run-time. A designer would think of this feature as a **Static Set of Product Types**.[1]

If a designer needs to have an **Open Set** of types handled by the pattern (as in, for example, vector<T> in STL where user of vector abstraction can specify and add to the existing pool of built-in C++ element types, i.e. int, char, double ...) we will see that there are several variants within the Factory Method design space that can help. The **Closed Set** indicates that the designer only requires a fixed number of product types to be handled (as in, for example, vectorInt, vectorChar, vectorDouble, ... all specialiazing vector type).

A particular client's request for creation of an object may be statically bound to a product type, or that choice may be delayed until run-time. The choice is captured by the notion of **Selection Time** and the **Static** or **Dynamic** alternatives.

The **creator** provides the factory method that returns the newly created product object. The **creator** role concept is not a top-level feature, as it does not concern the initial step of decision making, but it does enter into the picture in the choice between who makes a selection decision about the product type: the **client** that calls the factory method, or the **creator** that provides that factory method. The choice is captured by the notion of **Selection Place** and the **Client** or **Creator** alternatives.

2.3 Example: Design Variants For The Factory Method

In analyzing the Factory Method pattern we identify 8 distinct design variants, most of which are found in the implementation section. In the GoF format [3] the forces that would lead a designer to choose one variant over another are, at best, mentioned where the design space element is discussed, and at the worst, entirely omitted.

The core design idea in the Factory Method pattern is the **factory method** itself—a function that returns a newly created object to its **client**, decoupling its **client** from the specific type of **product**. The **factory method** is a delegate responsible for **creation**, and the **creator** class is a wrapper around that **factory method**. All the variants contain this core design idea in some form—this pattern's invariant is the factory method idea.

Now we explore the variants constituting the design space of the Factory Method pattern [3].

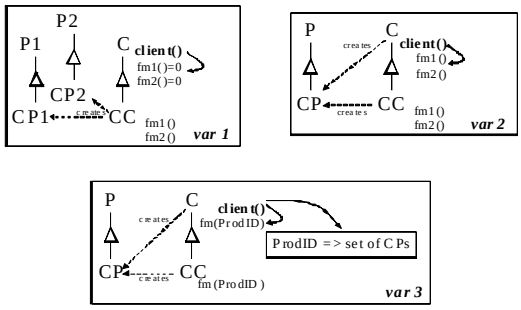


Figure 3. Variants 1-3

Variant-1 in Fig. 3, has the following elements:

P1, P2 — product interface for product category 1 and 2 respectively

CP1, CP2 — concrete product for product category 1 and 2 respectively

C — creator interface

fm1(), fm2() in C — factory method interface only

CC — concrete creator

fm1(), fm2() in CC — factory method implementation

client() — product client which is a template method within creator

Variant-1 elaborates the core idea by separating the factory method interface from its implementation. The different factory method implementations of that interface provide the mappings to the different concrete product types. There is a product hierarchy for each category of product: buttons and menus would be examples of categories and within these there are different types of buttons and menus, simple and fancy ones, for instance. This is modeled with P1, CP1 and P2, CP2 hierarchies corresponding to category-1 and category-2. The factory method interface in C, fm1 and fm2, map to category-1 and category-2 respectively, while the different concrete creators map to the concrete types within these categories. Client is typically a template method in the creator C itself, but the client can also be external to the creator. In subsequent variants we just model a single product category for the sake of not cluttering the diagrams. We do imply multiple categories by the existence of multiple factory methods.

The client is not simply a client of the product individually, but of the pattern as a whole: we can think of a "Factory Method" interface that a client has to follow to participate in the pattern contract. The basis of that contract

[1] Phrases or words in bold are concepts directly modeled in the corresponding feature model (Fig. 2).

is the client's assumption that there is a create-association from the creator hierarchy to the product class hierarchies. This is typical for design patterns: they present a contract-oriented interface to their clients.

Variant-2 in Fig. 3 has the same elements as Variant-1, except for the following difference:

fm1(), fm2() in C — factory method interface and *default* implementation

Variant-2 elaborates the core idea in much the same way as Variant-1, with the minor difference that fm1, fm2 in C provide *default* implementations. Hence a direct create association from creator C to a concrete product CP.

Variant-3 in Fig. 3 has the same elements as Variant-1, except for the following differences:

fm(ProdID) in C and CC — factory method interface and *default* implementation

ProdID=>set of CPs — a map between ProdID values and CPs (concrete product types)

Variant-3 elaborates the core idea by providing a parametrized factory method interface, and a mapping between the parameter values and concrete product types. This parametrized interface allows a single factory method to return multiple types of products, allowing the designer to replace all the different factory methods of variant-1 and variant-2 with a single factory method. In a statically typed language like C++ this means that fm(ProdID) will have a return type of Super-P, which will be a super-type for all the product categories. This may present a problem in that the client may want to invoke category-specific operations (menu specific operations, for example, as opposed to button specific ones). In that case the solution is to recover the type on client side through dynamic casting. In dynamically typed languages this is not an issue.

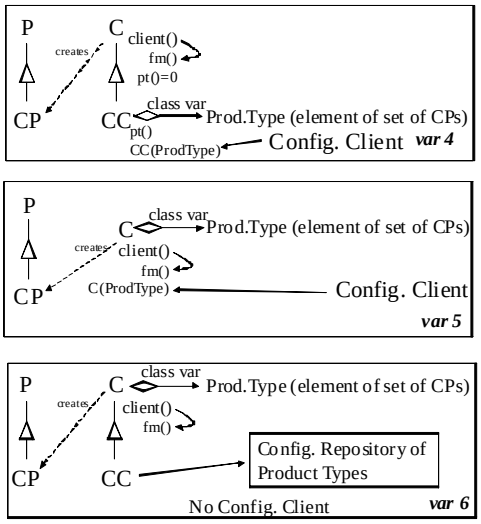


Figure 4. Variants 4-6

Variant-4 elaborates the core idea by having the factory method fm() delegate to CC, the concrete creator, the choice of CP. The factory method fm() in C is fully implemented but dependent on method pt() which returns the product type to be created. The method pt(), implemented in CC, simply returns type of product (not the product itself).

Variant-4 in Fig. 4 has the same elements as Variant-1, except for the following differences:

fm() in C only — factory method implementation, uses pt()

pt() in C and CC — method that returns type of product, with interface in C and implementation in CC

Config. Client, CC(ProdType) — Config. Client conFig.s CC with a particular type CP

ProdType — type variable with values from set of CPs

Variant-4 has a direct implementation in dynamically typed languages such as Smalltalk, but in statically typed languages, where types are typically not directly accessible at run-time, it is more appropriate to shift to a compile-time mechanism where types are visible. C++ templates are one such mechanism, although other meta-level mechanism are conceivable. Generally, when a design requires some kind of a type-computation, a dynamically-typed language would provide a direct implementation, whereas a statically-typed language would require use of a meta-level facility normally available only during compile-time. C++ does provide a run-time RTTI (Run-Time Type Identification) mechanism but that is both awkward and carries a performance penalty relative to a template based approach.

Variant-4 can be implemented using the Prototype pattern[3], where pt() would return a prototype object (rather than product type), on which the factory method fm() would invoke clone() operation.

Variant-5 in Fig. 4 differs from variant-4 in that it replaces method pt() with a class variable Prod. Type. Now, instead of subclassing creator C to provide different implementations of pt(), the creator C simply needs to be configured with a particular value for Prod. Type. The class variable eliminates the need for subclassing of C.

Variant-5 in Fig. 4 has the same elements as Variant-4, except for the following differences:

No CC, no pt() — there is only single creator class C; there is no method pt()

Config. Client, C(ProdType) — Config. Client configures C with a particular type CP

ProdType — type variable with values from set of CPs

Variant-6 in Fig. 4 is similar to Variant-5, but it eliminates the Config. Client by subclassing C and delegating to a concrete creator CC the selection of a value for Prod. Type from set of Product Types. The selection would take place in the constructor of CC.

Variant-6 in Fig. 4 has the same elements as Variant-4, except for the following differences:

No pt() —there is no method pt()

ProdType — type variable with values from set of CPs

Config. Repository of Product Types — repository of CP types

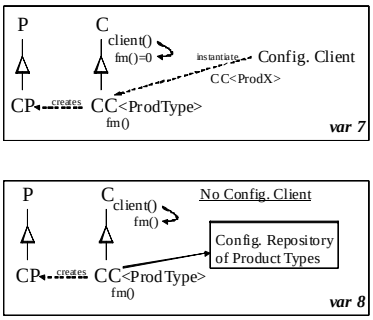


Figure 5. Variants 7-8

Variant-7 elaborates the core idea by type-parametrization of the concrete creator CC with the product type to be returned by the factory method fm(). This reduces the number of concrete creators to only one, regardless of the number of product types that need to be served by the factory method. The configuration client, Config. Client, passes the product type, ProdType, to the concrete creator.

Variant-7 in Fig. 5 has the same elements as Variant-1, except for the following differences:

CC — type-parametrized with the product type.

Config. Client, CC<ProdType> — Config. Client configures CC with a particular type CP

ProdType — type variable with values from set of CPs

Variant-8 in Fig. 5 is similar to Variant-7, but it eliminates the Config. Client by having a configuration repository which provides values for ProdType.

Variant-8 in Fig. 5 has the same elements as Variant-7, except for the following differences:

No Config. Client —there is no configuration client for CC

Config. Repository of Product Types — repository of CP types

With Variant-8 we conclude the description of the design space for the Factory Method pattern. From a designer's perspective the choice of inheritance, with and without templates, or parametrized factory methods, and other mechanisms are all refinements under the influence of forces modeled in Fig. 2. These refinements are aimed to have a direct effect on the **factory method**, and only an indirect one on the **creator**. A purely functional version of this pattern can be envisioned, where there is no creator class, but only a free-standing factory method.

The core idea is a seed out of which the variants emerge to form the design space. This is the basis for the generative process driven by the forces in Fig. 2. All the variants, therefore, contain this design idea in some form. We could envision the core idea of this design pattern as being selected, and then refined and adapted by the designer under the forces modeled in Fig. 2.

2.4 Example: Mapping Design Forces To Design Variants For The Factory Method Pattern

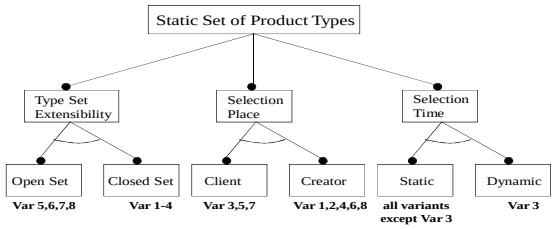


Figure 6. Factory Method Feature Model to Design Space Mapping

Fig. 6 illustrates a mapping from a model of forces influencing a designer to variants provided by the Factory Method pattern. As the feature model indicates, there are a lot more choices left once the pattern itself is chosen: Is the set of product types open or closed? Who makes the selection and when? Is there a dependence between product types that needs to be addressed at creation time? Analysis of these questions and their answers gives rise to the feature model (Fig. 2), variants constituting the design space (Fig. 3-5), and most significantly, a mapping from one to the other (Fig. 6).

An **Open Set** feature indicates that the designer needs to allow for expansion in the set of product types served to the client. The approach used is to externalize, through parametrization, the configuration of creator with the product type. Variants 5,6 make that externalized product type a run-time parameter which makes them more suitable for dynamically typed languages (eg. Smalltalk), while Variants-7,8 make them a compile-time parameter through the mechanism of type-parametrization (generic types). Clearly, Variants-7,8 imply a language like C++ with its template mechanism.

Variant-3 in Fig. 3 is not considered to support the **Open Set** feature because the mapping of ProdID to concrete product types is implemented programmatically inside the factory method fm(ProdID) as illustrated by the following code (p111 [3]) of:

```
class Creator {
        public:
           virtual Product* fm(ProductID);
        };

        Product* Creator::fm (ProductID id) {
           if (id == MINE)  return new MyProduct;
           if (id == YOURS) return new YourProduct;
           // repeat for remaining products...

           return 0;
        }

Product* MyCreator::fm (ProductID id) {
```

```
    if (id == YOURS)  return new MyProduct;
    if (id == MINE)   return new YourProduct;
      // N.B.: switched YOURS and MINE

    if (id == THEIRS) return new TheirProduct;

    return Creator::fm(id); // called if all others fail
}
```

A **Closed Set** feature indicates that set of product types is not extensible, and its choice will lead to Variants-1-4 (fig. 3-5) part of the design space. These variants are closed with respect to set of product types because they hardwire that in the definition of the concrete **factory methods**, as opposed to Variants-5,7 (Fig. 4 and Fig. 5) which provide for a configuration interface.

Who makes the selection and when? Is the selection (mapping of factory method to product type) made by the client or by the creator? Is it made at the time creator is defined, or when it is initialized, or when the client requests a product object?

"Who makes the selection?" is also a question of the interface the pattern presents to the client. Does the client need to make the selection? If the client needs to control the mapping, i.e. **Selection Place->Client**, then designer is led to those variants that externalize the mapping: Variants-3,5,7 (Fig. 3-5). If the client does not need to control that mapping, i.e. **Selection Place->Creator**, then Variants 1,2,4,6,8 (Fig. 3-5) would be designer's choices. In Variant-3 (Fig. 3), as seen in the code shown above, the definition of the mapping is hidden from the client, but the selection based on that mapping is available to the client. Hence, an explicit element in Variant-3 that shows the ProdID=>CP map implied by the fm() implementation. The intent in this variant is to replace N factory methods with a single one: fm1(), fm2() ... fmN() as a mechanism for allowing client choice of product categories 1,2 ... N (not choices of concrete product types for which the inheritance mechanisms is used) is replaced by fm(ProdID). More generally, the map can be envisioned as a way to externalize the relationship of fm(ProdID) to concrete product types and categories, and thereby provide for a reconfigurable way of mapping client creation requests to product types and categories. In this case the factory method fm(ProdID) could simply dispatch on ProdID using the shared ProdID=>CP map.

"When is the selection made?" has an answer that is dependent on the implementation language used. In a statically typed languages like C++ and Java, only Variant-3 (Fig. 3) offers **Selection Time** that is **Dynamic**; the rest of the variants are all **Static**, with a possibility of "simulating" dynamic selection for Variant-5 if an additional dynamic configuration interface is added (eg. changeProdType(prototype)) that complements the existing C(ProdType) interface. The "simulation" is in using a change in prototype object to simulate change in actual product type. This level of language-dependent detail could be modeled as further refinements in the Feature Model with corresponding variants in the design space. The point is that the Feature Model captures the decision making information in layers, with upper layers modeling architectural and design level decisions and lower ones, the detailed-design and implementation level decisions.

Other Creational Patterns

As we have already mentioned the core design idea in the Factory Method pattern is the **factory method** itself—a function that returns a newly created object to its **client**, decoupling its **client** from the specific type of **product**. The **factory method** is a delegate responsible for **creation**. This core idea is shared to some extent with all the creational patterns in [3]. Factory Method puts the creator role inside the client itself, making the client a template method within an existing class (eg. CreateMaze(), a template method, is a member function of the MazeGame class p114[3]). Abstract Factory separates the creator from the client completely, thereby allowing for dynamic selection of creator. Prototype makes things even more flexible: the creator (factory method) can be configured at run time. Builder is about building arbitrary structures rather than individual products and singleton defines a kind of factory method that ensures uniqueness.

3. Conclusion

Variability

The Factory Method analysis illustrates in some detail that the core pattern conceptualization – a design solution to a problem in context – is better described as a family of design solutions to a family of problems in a family of contexts: variability within and across each component of a pattern is fundamental. In [17] we investigate potential modeling solutions to deal with variability.

Even a basic and fundamental pattern like the Factory Method has significant complexity: Fig. 2 models a family of design problems, Fig. 3-5 model a family of design solutions, and Fig. 6 models a mapping from problem family to the solution family.

Navigation and Selection

The variants found in each pattern are points in some design subspace: the 8 variants of the Factory Method pattern define a Factory Method subspace in the larger Creational pattern design space. We can imagine the designer as a navigator in that space, moving towards one pattern subspace over another under the influence of forces, and selecting a particular point, i.e. variant, in that space. In these terms, the GoF format provides a first-level, sparse description of both the space defined by a particular pattern, and the forces acting upon a designer to select a pattern.

Evaluating Choices

How do we evaluate that the chosen variant satisfies the specific goal and that the implementation satisfies the properties of the design?

We are currently developing a technique based on the use of Another Contract Language (ACL) as a pattern modeling language where the corresponding implementations are evaluated using the Validation Framework [14].

Our research goals are to deal with these challenges:

- Traceability [18] from goals and intents to variations and their implementations,
- Variability modeling in analysis, design, and implementation, [17]
- Interaction between traceability and variability, [17] and
- Evaluation of selected variants relative to intents and detailed design properties, and corresponding implementations, [12] and
- Development of tools that would assist designers in evaluating implementations. [12]

4. Acknowledgments

Support from the Natural Science and Engineering Research council of Canada is gratefully acknowledged.

5. References

[1] C. Alexander, S. Ishikawa, M. Silverstein, with M. Jacobson, I. Fiksdahl-King, & S. Angel, *A Pattern Language* (New York, NY: Oxford University Press, 1977).

[2] C. Alexander, *The Timeless Way of Building* (New York, NY: Oxford University Press, 1979).

[3] E. Gamma, R. Helm, R. Johnson, & J. Vlissides, *Design Patterns-Elements of Reusable Object-Oriented Software* (Reading, MA: Addison-Wesley, 1994).

[4] F. Buschman, & K. Henney, A Distributed Pattern Language. Proc. Seventh European Conference on Pattern Languages of Programs, Irsee, Germany, 2002.

[5] W. Pree, *Design Patterns for Object-Oriented Software Development* (Reading, MA: Addison-Wesley, 1994).

[6] M. Fontoura, W. Pree, & B. Rumpe, *The UML Profile for Framework Architectures* (Reading, MA: Addison-Wesley, 2002).

[7] K. Czarnecki, & U.W. Eisenecker, *Generative Programming: Methods, Tools, and Application,* (Reading, MA: Addison-Wesley, 2000).

[8] F. Buschman, R. Meunier, H. Rohnert, P. Sommerlad, & M. Stall, *Pattern-Oriented Software Architecture - A System of Patterns* (Chichester, West Sussex, England: Wiley and Sons, 1996).

[9] D.C. Schmidt, & S.D. Huston, S.D., *C++ Network Programming: Mastering Complexity With ACE and patterns* (Reading, MA: Addison-Wesley, 2002).

[10] A. Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied,* (Reading, MA: Addison-Wesley, 2001).

[11] K. de Volder, *Implementing Design Patterns as Declarative Code Generators*, 2001- http://www.cs.ubc.ca/~kdvolder/

[12] V.D. Radonjic, *An Open Pattern-based Framework for Evaluating Software Systems*, PhD. Thesis (in preparation) Carleton University, School of Computer Science.

[13] K. Beck, & R. Johnson, *Patterns Generate Architectures*. Proc. ECOOP'94, Bologna, Italy,1994.

[14] D. Arnold, *Validation Framework and Another Contract Language*, http://vf.davearnold.ca/

[16] M. Antkiewicz, K. Czarnecki, and M. Stephan, *Engineering of Framework-Specific Modeling Languages*, IEEE Trans. Software Eng., vol. 35, no. 6, pp. 795-824, Nov/Dec 2009.

[17] S. Bashardoust, V.D. Radonjic and J.-P. Corriveau and D. Arnold, *Challenges of Variability in Model-Driven and Transformational Approaches: A Systematic Survey*, Workshop on Variability in Software Architecture, WICSA2011, Boulder, Colorado, USA, 2011.

[18] J.-P. Corriveau, *Traceability Process for Large OO Projects*, IEEE Computer, pp. 63-68, Sep 1996.

On the Exploration of Lightweight Reverse Engineering Tool Development for C++ Programs

Yan Liang
Computer Science Department, the University of Alabama, Tuscaloosa, AL, USA

Abstract - *Reverse engineering (RE) is the process that examines a software system and creates a higher level of abstraction of representation for the system. Building reverse engineering tools is expensive because of the complexity of programming languages and the iterative nature of tool development process. This paper explores the reuse of existing technologies and tools to develop a RE tool as Eclipse plug-in for C++ programs. We use CDT API to extract artifacts from C++ source code, avoiding the burden of complicated parsing process. To better handle model creation, we develop our tool in a model-driven development (MDD) environment - Eclipse Modeling Framework (EMF), taking its advantages on modeling, automatic code generation, constraint imposing and validation, and serialization. We discuss how this development approach could affect the quality of the RE tool in terms of performance and accuracy. Our experience shows that the reuse of existing extraction component together with model-driven methodology for RE tool development can reduce the cost of RE tool development.*

Keywords: Reverse Engineering, Model-Driven Development, Meta-model, Abstract Syntax Tree

1 Introduction

With the tremendous increase of the amount and complexity of software systems, tool-based source code introspection has become a necessity to support software maintenance. Reverse engineering (RE) technologies analyze a subject system, identify its components and their interrelationships, and create representations of the system in another form or at a higher level of abstraction [1]. RE tools have gained growing interest in the last 20 years due to their successful contributions on helping software engineers better understand the system. Application domains of RE technologies have been expanded and brought benefits to more areas such as re-documenting programs, recovering architectures, clone detection, change impact analysis and recovering design patterns [12]. The powerful capabilities of RE technologies in these areas have practically reduced the human efforts on the maintenance of software legacy systems.

Source code gains consistent interest among all of software artifacts because it is the one that actually implements application rules. When a software system has poor documentation, reverse engineering of source code becomes the focal point to enhance the maintainability of the system. Generally, a RE tool has three fundamental components [2]. An extractor parses code, extracts code artifacts and builds abstract syntax tree (AST) or abstract semantic graph (ASG) for input programs. An abstractor then analyzes the extracted artifacts and changes them into higher level representation. This representation can be used for further code analysis or visualization within the tool. A visualizer displays the output of abstractor in a graphical way.

Corresponding to three functional components of a RE tool are three levels of meta-models. The extractor needs an AST-level meta-model which includes complete syntactic and semantic information. The abstractor operates on a refined meta-model that captures certain program features in an abstract form congruent with the needs of RE tasks. The visualizer requires a graphical structure for display. These meta-models may vary dramatically among different RE tools.

2 Cost of Building RE Tools

Building a RE tool is expensive. One reason is the intricacy of constructing a fact extractor. Some tools build their own parsers. Others leverage compiler frontends to fit the demands of tasks that RE tools intend to undertake [5]. However, both are still time-consuming. In addition, RE tool development, like other software development activities, is an iterative process. A particular example of its iterative nature occurs between the design of the target meta-model and the execution algorithm of meta-model instantiation. Changes on the meta-model must be followed by corresponding modifications on code and its computation logic. Cost of development may increase dramatically with every time the meta-model changes. This is particularly true for RE tool development since its working objects are programming languages with complicated lexical, syntactical and semantic features.

This concern drives RE researchers and practitioners seeking new techniques to improve traditional tool development methodologies that construct everything from scratch. The basic idea is building RE tools upon existing functionality and infrastructure. Recently, two major techniques have been proposed: Component-Based Tool Development (CBTD) and Model-Driven Tool Development (MDTD). CBTD emphasizes the reuse of existing code in the form of component. MDTD moves the level of abstraction from implementation details to model design with intent to solve issues code-driven development generally has. Kienle and Muller name CBTD and MDTD as compositional and generative reuse [13].

In this study, we introduce a RE tool development process that merges the ideas of CBTD and MDTD. We use an AST API to extract artifacts from source code, avoiding the burden

of complicated parsing process. To better handle target models, we choose a model-driven development (MDD) environment to utilize its versatile functionalities of model management. Two parts constitutes our development environment to implement a source-code-to-model transformation.

The remainder of this paper is organized as follows. In section 3 we introduce development tools and the workflow under our development environment. In section 4 we present details and results at different execution stages. Section 5 discusses the issues relevant to the tools and the approach we adopt. Related work is presented in section 6 followed by our conclusion in section 7.

3 Lightweight Tool Development

To support continuous reverse engineering, a new trend is incorporating RE tools into integrated development environment (IDE). This enables developers to reverse engineer their programs while coding. In this section, we first introduce tools we adopt to develop an Eclipse plug-in that reverse engineers C++ source code into a representation conforming to a pre-defined meta-model. We then present the working process that injects MDD ideas to accomplish our development goal.

3.1 EMF

Eclipse Modeling Framework (EMF) is the fundamental part of Eclipse modeling project and provides basic framework for modeling in Eclipse IDE. Given the Ecore model of a domain application, EMF framework and code generation facility can generate XML, UML and Java code to represent and manipulate the domain meta-model [4].

This project employs EMF for:

- Modeling: the concrete application meta-model is modeled as an instance of EMF Ecore meta-model. Ecore meta-model is the core of EMF, and is designed to map cleanly to Java implementation. A prominent feature of EMF is allowing direct constraint specification by attaching expressions written in Object Constraint Language (OCL) as annotations to Ecore model elements. EMP provides extensible code generator that can convert OCL expressions to Java code for model validation. In this way, constraint imposing and meta-model design are seamlessly integrated in EMF.
- Code generation: Java code for model implementation is generated based on the meta-model defined. The generated code has two basic packages: interface package contains the set of Java interface representing the client interface to the model; implementation package contains corresponding implementation classes. Design patterns such as adapter pattern and factory method pattern are applied to the generated code for the enhancement of code reusability and performance. Other useful packages adhere to the meta-model can also be generated such as utility package and validation package.
- XML serialization: EMF facilitates a highly customizable resource implementation in support of model object persistence.

3.2 CDT API

While Eclipse framework is implemented using Java, it has been used to implement development tools for other languages. CDT is a C++ IDE based on Eclipse platform. It also provides API to introspect C/C++ code [8]. CDT uses three different models to organize the syntactical and semantic information of C/C++ code:

- C-Model intends to populate project navigator and outline view without local declarations included.
- C-Index captures each name and binding. It works as connections between declarations and references.
- Abstract Syntax Tree (AST) contains every detail about code such as scope, preprocessor directives, return types and parameters of functions, macro expansions, comments and types of variables.

CDT API provides client programs with access to multi-view representations of C/C++ code to handle the scalability issue of model creation. Retrieval of C/C++ source code artifacts via CDT API is expected to perform efficiently by accessing proper models.

3.3 Development Workflow

Figure 1 shows our development process using Eclipse EMF and CDT API. It starts with creating a meta-model as .ecore file for a specific RE task working on C++ programs. On top of .ecore file a code generation model with file extension .genmodel is created to describe code generation pattern that does not belong to Ecore model itself but affects the code to be generated. Based on .genmodel file, EMF generator produces corresponsive Java source code. Code customization may be conducted to add new methods or rewrite generated methods when necessary.

When the code for manipulating the meta-model is readily available, we extract language elements from C++ source code via CDT API, and map them into model elements. In the last step, the mapping result is serialized into an XMI file as the final output.

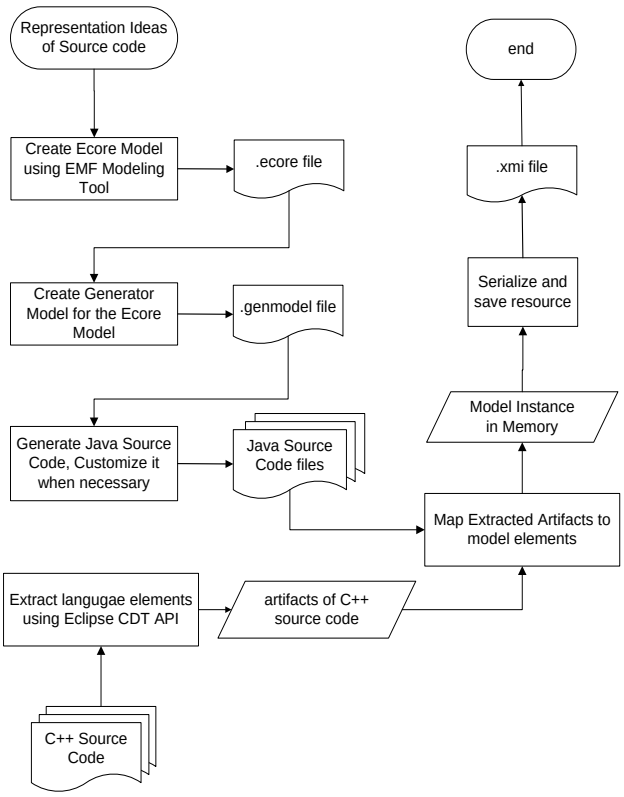


Figure 1: Development Workflow

4 Execution

In this section, we present execution results obtained in each step of our development workflow. The development goal is to extract classes and their members defined in a C++ project. These language elements are commonly used in lots of object-oriented analysis tasks such as class hierarchy analysis and accessibility analysis. All the results are produced in the environment composed of Eclipse 3.6, EMF 2.6, MDT/OCL 3.0 and CDT 7.0.

4.1 Create Domain Meta-model

There are several tools available in Eclipse to create Ecore models. This development project uses the simple tree-based sample editor supplied by EMF and OCLinEcore text editor by Eclipse Model Development Tool (MDT) project.

Figure 2 shows the UML representation of our domain meta-model. On the top-right are two enumerations. *CPPAccessSpecifer* specifies three member access levels. *CPPClassKind* makes distinctions among C++ declarations of class, struct and union. This is essential for accessibility analysis because the default access specifier of a C++ class member is private, whereas members of a structure or union are public by default. Classes *CPPID* and *CPPNamedElement* are abstract classes that are inherited by most of model classes. Therefore, each concrete class has an integer as its identifier. Except for *CPPInheritance*, all other concrete classes are named classes. *CPPInheritance* represents the inheritance relation between a base class and its derived class.

All concrete classes are organized hierarchically by composition edges. The root of this hierarchy is a *CPPProject* class, which contains multiple *CPPClass* objects. *CPPClass* has the most complicated properties among all model classes. A *CPPClass* object may contain instances of *CPPInheritance*, *CPPDataMember*, and *CPPFunctionMember*. It may also enclose other objects of *CPPClass* and *CPPMemberFunction* that correspond to the declaration of nested classes, friend classes or friend functions in C++ source code.

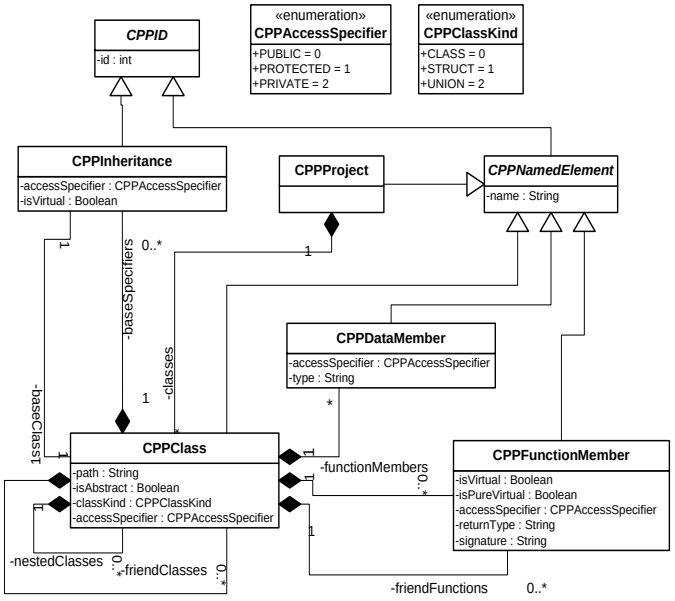


Figure 2: UML Representation of Meta-model

4.2 Generate Java code

Based on the meta-model stated above, a generator model is created to drive Java code generation. To transform OCL constraints to the source code for execution, we modify some properties of the generator model that are created by default.

The generated code supplies a variety of functionalities to support different operation requirements on the domain meta-model. For example, each classifier defined in the Ecore model is translated into an interface and an implementation class. A factory interface and its implementation are generated for each model package to create model objects. A validator class is also generated which includes a validate method for each classifier defined in the model package. Figure 3 is the generated code demonstrating these features. Figure 3(a) is the interface Generated for *CPPClass*, (b) is the factory interface, and (c) is a validate method for the OCL constraint defined in the context of *CPPClass*, which appears as one of inclusive methods of the validator class in the meta-model package. In Eclipse OCL 3.0, OCL expressions appear in the generated code as string. In the presented snippet, the `notNestedSelf`

OCL expression attached to the meta-model definition is stored in CPP_CLASS__NOT_NESTED_SELF__EEXPRESSION, and will be interpreted later in our program.

We perform a few customizations on the generated code. As an example, according to the meta-model, we need an integer as project-wide identifier for each model object.

```
public interface CPPClass extends CPPNamedElement {
    String getPath();
    void setPath(String value);
    boolean isIsAbstract();
    void setIsAbstract(boolean value);
    CPPClassKind getClassKind();
    void setClassKind(CPPClassKind value);
    CPPAccessSpecifier getAccessSpecifier();
    void setAccessSpecifier(CPPAccessSpecifier value);
    EList<CPPClass> getNestedClasses();
    EList<CPPClass> getFriendClasses();
    EList<CPPInheritance> getBaseSpecifiers();
    EList<CPPDataMember> getDataMembers();
    EList<CPPFunctionMember> getFunctionMembers();
    EList<CPPFunctionMember> getFriendFunctions();
}
```

(a) Interface Generated for *CPPClass*

```
public interface ModelExampleFactory extends EFactory {
    ModelExampleFactory eINSTANCE = model
    Example.impl.ModelExampleFactoryImpl.init();
    CPPClass createCPPClass();
    CPPDataMember createCPPDataMember();
    CPPInheritance createCPPInheritance();
    CPPFunctionMember createCPPFunctionMember();
    CPPParameter createCPPParameter();
    ModelExamplePackage getModelExamplePackage();
}
```

(b) Factory Interface

```
public boolean validateCPPClass_notNestedSelf(CPPClass
cppClass, DiagnosticChain diagnostics, Map<Object, Object>
context) {
 return validate
  (ModelExamplePackage.Literals.CPP_CLASS,
    cppClass, diagnostics, context,
                "http://www.eclipse.org/emf/2002/Ecore/OCL",
                                "notNestedSelf",
          CPP_CLASS__NOT_NESTED_SELF__EEXPRESSION,
   Diagnostic.ERROR, DIAGNOSTIC_SOURCE, 0);
 }
```

(c) Validation Method for OCL Constraint

Figure 3: Snippet of Generated Java Code

4.3 Map Program Artifacts to Model Elements

The mapping process is conducted by working on the generated Java source code. To start a model construction process for C++ programs under analysis, an *ICProject* class object is first extracted from CDT C-Model and mapped into *CPPProject* model class object. Other mappings are accomplished by using visitor pattern for multiple traversals on different objects of CDT C-Model and AST model. In detail, mappings are implemented through three AST visitors we create: class abstractor, inheritance abstractor and friend abstractor. Brief introductions about these abstractors are listed below. To build the inheritance abstractor and friend abstractor, class abstractor is created in the first place to extract all classes and their members from source code.

- Class Abstractor: create *CPPClass* objects for all C++ classes defined in source code (including nested classes) by visiting IStructure class of CDT C-Model. In the meantime, by accessing *IField* and *IMethod* classes of C-Model, we can also create objects for model classes *CPPDataMember* and *CPPFunctionMember* accordingly, along with the containment relations to their containing *CPPClass* objects.
- Inheritance Abstractor: create *CPPInheritance* model class objects by visiting *IASTName* objects of CDT AST and using indexes. Each *CPPInheritance* object is contained in a *CPPClass* object that corresponds to a derived class in the C++ source file under analysis.
- Friend Abstractor: identify friend classes and friend functions to complete information each *CPPClass* object needs to obtain. Similar to inheritance extraction, this abstractor uses information from CDT AST and indexes.

Each abstractor iterates all translation units (TU) and fills up the output model in an incremental way. Figure 4 is the algorithm for the inheritance abstractor.

```
For each TU{
   Create AST, skipping function bodies
   For each C++ class name astDClass in AST {
        Get base class(es) astBClass
     Find matching modeDlClass for astDClass
     If modelDClass has not been processed {
       Find matching modelClass for astBClass
        Create CPPInheritance object
          Add CPPInheritance object into modelDClass
       }
   }
}
```

Figure 4: Algorithm for Inheritance Abstractor

4.4 Serialization

The serialization of all model objects is quite simple with EMF persistence framework when all objects are organized as a tree rooted at the *CPPProject* object. Adding the *CPPProject* object to the EMF resource's list of content and calling the save() method will serialize all objects of the generated model resident in memory.

Figure 6 is the screenshot generated from a serialized model. Figure 5 is the source code corresponding to this model that demonstrates the usage of observer pattern in C++ [9]. The screenshot has two separate regions. On the top is the model structure of the input program. A *CPPProject* object appears as root to populate a complete model. On the bottom is a window showing the properties of a *CPPFunctionMember* class object named AnalogClock.

```
class Subject;
class Observer {
 public:
     virtual  ~Observer(){};
      virtual void update(const Subject& theChangedSubject) = 0;
 protected:
     Observer(){};
 };
class Subject {
 public:
     virtual ~Subject();
  virtual void attach(Observer*);
     virtual void detach(Observer*);
     virtual bool notify();
 protected:
      Subject(); // protected default ctor
 private:
      typedef std::list<Observer*> ObsList;
      typedef ObsList::iterator ObsListIter;
      ObsList mObservers;
 };
class ClockTimer : public Subject{
  public:
      ClockTimer(){};
      virtual int getHour()const;
      virtual int getMinute() const;
    virtual int getSecond() const;
    void tick();
};
class Widget { /* ... */ };
class DigitalClock: public Widget, public Observer{
 public:
      explicit DigitalClock(ClockTimer&);
      virtual ~DigitalClock();
      virtual void update(const Subject&);
     virtual void draw();
 private:
      ClockTimer& mSubject;
};
class AnalogClock : public Widget, public Observer{
 public:
      AnalogClock(ClockTimer&);
      virtual void update(const Subject&);
      virtual void draw();
};
```

5 Discussion

In general, the adoption of EMF and CDT API speeds our development process. Meanwhile, we feel that this approach also has impact on a number of generic quality attributes that RE tools should strive to meet. In this section, we discuss the features of EMF and CDT API that affect the performance and accuracy of a RE tool.

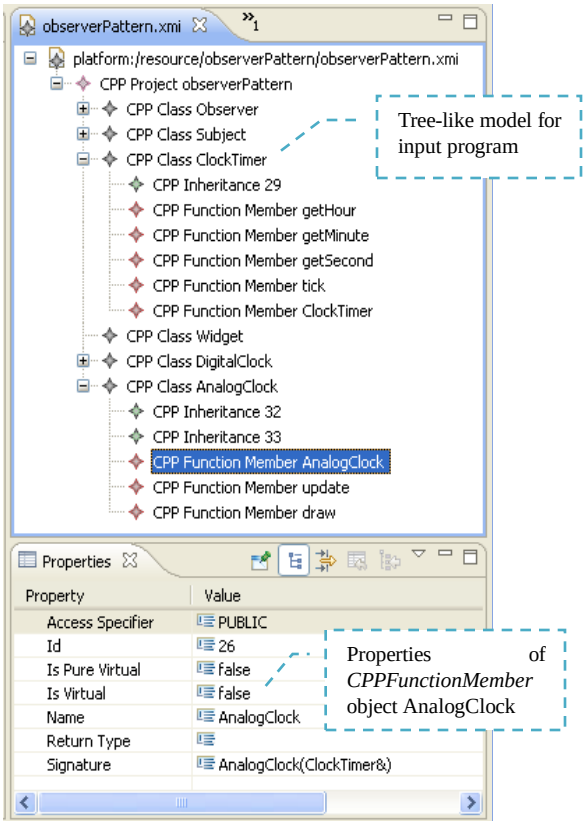


Figure 6: Screenshot of the Output Model

5.1 Usage of EMF

In this study, the target meta-model is explicitly defined. However, the source meta-model, that is, the structure of AST for C++, is hidden behind API. Therefore, meta-model transformation is realized by using general-purpose programming language (Java in this development). This is different from the typical MDD paradigm where both the source and target meta-models are visible to developers and model transformation is implemented by model transformation language such as ATL [7]. An alternative approach for RE tool development is using AST meta-model that is explicitly specified for direct use. This approach is called model-driven reverse engineering. A representative project applying this idea is Eclipse Modisco, which defines AST-level meta-model for Java programs [15]. COLUMBUS project provides AST meta-model for C++ programs for RE and reengineering tasks [16]. In our opinion, both approaches have advantages and disadvantages. The overall tool performance could be better if the execution is based on APIs. However, developers need more programming skills and spend more time on execution details. Explicit meta-model eases the meta-model transformation process but may compromise tool performance.

The Java code generated automatically by EMF highlights one of the most important benefits from MDD. Generated code incorporates good coding and design practice to enable reusability and adaptability for client programming. For example, parameterized factory method pattern is used in the generated code to separate all other functional implementation from how objects are created, composed and represented [9]. During the development, we change the design of target meta-model several times. Due to MDD approach, we don't have too much trouble to adapt to these changes.

The validation framework of EMF provides with us a shortcut to examine whether the generated model contains errors, oversights or bugs. EMF allows us to expand and customize the verification rules through user-defined OCL constraints. These constraints are attached to the meta-model and can be translated into Java code directly. As a result, developers can invoke validation process during or after the model creation. Violation of constraints can be detected immediately without extra effort of programming.

5.2 Usage of CDT API

Generating AST for each translation unit (TU) is very time-consuming. CDT API provides two ways to solve this issue. First is the use of C-Model. In our example, ASTs are used for inheritance and friend abstractor since they need information of name resolution supplied by AST-level model. For class abstractor, information from C-Model is sufficient. Second, CDT API allows selectively generating AST for each TU. For instance, we choose to generate ASTs without function and method body, which is reasonable according to the design of our target meta-model. ASTs can also be generated by skipping head files included in C++ programs. Standard libraries such as stdio.h are among the best candidates that can be ignored.

The use of CDT API increases the flexibility of data extraction. However, this also means that the overall quality of our RE tool would be limited to the functions these APIs can provide. CDT API helps us achieve the development goal. However, to support more complicated RE applications, more work is required to measure its capabilities with respect to semantic completeness, robustness, performance and accuracy.

6 Related Work

A great deal of publicity on reverse engineering has appeared in the past two decades. Canfora et al. reviews important achievements and areas of application, and present concerns and future research directions [12]. Kienle and Muller have comprehensive discussions on RE tool development in terms of requirements, software architectures and tool evaluation criteria [13]. There are many other studies focusing on RE modelling issues [2][14][16].

Research on RE tools has a long history. However, projects applying the idea of MDD on RE tool development are very limited. Rugaber and Stirewalt presented their study on model-driven RE. Their work is motivated by the expectations of project managers on RE and intends to enable better development effort prediction and quality evaluation of RE [10]. Another example is model-driven visualization for program comprehension by Favre et.al [11]. Its source meta-model adheres to the Dagstuhl Middle Model, the visualization meta-models are described with Emfatic/EMF, and the transformations are written with ATL. MoDisco is an Eclipse GMT project for model-driven reverse engineering. By making use of other Eclipse solutions such as EMF, Modisco aims to provide an extensible and generic framework for model discovery and understanding [15]. It consists of multiple reusable components: meta-models to describe existing systems, discoverers to create models automatically, and generic tools to understand and transform complex models created out of existing systems. Modisco processes Java programs and cannot model C++ programs currently.

Model-driven development and model-driven engineering have long been reported as promising approaches to handle the issues of code-centric development. Model-driven tools and methodologies in software projects can simplify development work and automate endless change cycles. In addition to EMF we use in this study, other MDD environments have been developed. GME is a generic modeling environment developed at Vanderbilt University. SpecExplorer is an integrated environment for MDD of .NET software. IBM Rational® Rhapsody® Developer is a UML/SysML-based, model-driven development environment for real-time and embedded systems and software [6].

In addition to API, an alternative approach to code introspection is navigating AST-like models that include complete language entities and are ready for all kinds of analysis tasks. To automatically generate such a model, language grammar is associated with AST-level meta-model to generate a parser. EMFTextEdit is such a tool that can automatically derive an Ecore model, a lexer and parser from the abstract and concrete syntax definition of a language [17]. Textual Concrete Syntax (TCS) is a domain-specific language (DSL) that can express specifications to bridge abstract and concrete syntaxes for languages, and automatically generate tools for model-to-text and text-to-model transformation [3]. In our opinion, grammar based model generation is more suitable for DSLs because their grammars are closely related to domain concepts. For general purpose programming languages such as C++ and Java, this may not be a good idea. For C++, one of other reasons is that its macro and preprocessor directives are agnostic to its grammar.

7 Conclusion

We present our tool development practice for reverse engineering C++ source code. Two primary tools we adopt for the implementation are Eclipse EMF and CDT API, which stand out two prominent features of this development. First is the reuse of existing parser and its API to access source code entities, avoiding the burden work on parser creation. The second is the adoption of MDD environment to facilitate the design and realization of the target model as output. This also

allows us to reuse serializer, visualizer and model validator to complete essential functions a RE tool should contain. Our experience shows that the collaboration of EMF and CDT API makes our development more productive.

We present issues we have in the study. In particular, we discuss how the adopted tools could affect the performance and accuracy of the RE tool being built. We hope this practice is helpful to similar tool development tasks.

8 References

[1] E.J.Chikofsky, J.H. Cross. Reverse engineering and design recovery: A Taxonomy. IEEE Software. 7(1) (1990) 13–17.

[2] Richard C. Holt, Andy Schürr, Susan Elliott Sim, Andreas Winter. GXL: A graph-based standard exchange format for reengineering‖, Science of Computer Programming, volume 60, Issue 2, April 2006, Pages 149-170.

[3] F. Jouault, J. B´ezivin, and I. Kurtev. TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering. In Procs. Fifth Intl. Conference on Generative Programming and Component Engineering (GPCE'06), 2006.

[4] Dave Steinberg, Frank Budinsky. Marcelo Paternostro and Ed Merks, EMF: Eclipse Modeling Framework (2nd Edition), ISBN 0-321-33188-5, Addison-Wesley, 2008.

[5] N. A. Kraft, B. A. Malloy, and J. F. Power. g4re: Harnessing gcc to reverse engineer C++ applications. In Seminar No. 05161: Transformation Techniques in Software Engineering, Schloss Dagstuhl, Germany, April 17-22 2005..

[6] Douglas C. Schmidt, "Guest Editor's Introduction: Model-Driven Engineering," *Computer*, vol. 39, no. 2, pp. 25-31, Feb. 2006, doi:10.1109/MC.2006.58

[7] http://www.eclipse.org

[8] Markus Schorn, "Using CDT APIs to programmatically introspect C/C++ code". Eclipsecon 2009.

[9] Eric Gamma, Richard Helm, Ralph Johnson and John M. Vlissides: Design Patterns: Elements of Reusable Object-Oriented Software. ISBN 0-201-63361-2.

[10] S. Rugaber and R. E. K. Stirewalt. Model-driven reverse engineering. IEEE Software, 21(4):45–53, July/August 2004.

[11] J.Favre. Cacophony: Metamodel-driven software architecture reconstruction, 11th IEEE Working Conference on Reverse Engineering (WCRE) 2004.

[12] Gerardo CanforaHarman and Massimiliano Di Penta. New frontiers of reverse engineering. In FOSE'07: 2007 Future of Software Engineering, pages 326-341, Washington, DC, USA, 2007. IEEE Computer Society.

[13] Holger M. Kienle and Hausi A. Muller. The Tools Perspective on Software Reverse Engineering: Requirements, Construction and Evaluation. Advances in Computers. volumn 79. 2010. Pages 189-290

[14] N.A. Kraft, B.A. Malloy and J.F. Power, ―An Infrastructure to Support Interoperability among Reverse Engineering Tools,‖ Information and Software Technology, 49(3):292-307, March 2007.

[15] Hugo Bruneliere, Jordi Cabot, Frédéric Jouault and Frédéric Madiot. MoDisco. A Generic And Extensible Framework For Model Driven Reverse Engineering, ASE'10, September 20–24, 2010, Antwerp, Belgium

[16] R. Ferenc, S.E. Sim, R.C. Holt, R. Koschke and T. Gyimóthy. Towards a standard schema for C/C++. In: Working Conference on Reverse Engineering—WCRE 2001 (Stuttgart, Germany) (2001), pp. 49–58.

[17] http://emftextedit.reuseware.org.

An Approach for Generating Architectures for Pervasive Systems from Selected Features

Mostafa Hamza, Sherif G. Aly, Hoda Hosny
Department of Computer Science and Engineering
The American University in Cairo, Cairo, Egypt
{hamza, sgamal, hhosny}@aucegypt.edu

Abstract - *In this paper, we present a feature-driven approach for generating pervasive systems architectures. Our approach is based on the methodology for automatic generation of pervasive systems architectures from a predetermined architectural feature set. A set of pervasive systems architectural features relevant to such kinds of systems was researched, and a mapping between such features and relevant component diagrams was established. Based on the selection of features, an automatically generated architecture is constructed through the aggregation of various kinds of components relevant to the selected features. This effort is a first attempt towards the creation of a configurable reference architecture for pervasive systems, and the subsequent specification of a product line supporting the rapid development of such types of systems. Using the Feature Modeling Plug-in (FMP) with Eclipse and the Visual Paradigm for UML, we developed a program in C# that generates pervasive systems architectures. We used FMP with Eclipse in order to generate a feature model for pervasive systems under design. Visual Paradigm for UML was then used to map the different selected features to component diagrams. The component diagrams were then aggregated using known design patterns to generate the architecture in question.*

Key words: pervasive, architectural features, reference architecture, component-based architecture.

1. Introduction

Pervasive systems are thought to be the third wave of computing, interbreeding many disciplines in the computing domain. In specific, it is signified by the merger between mobile and distributed systems. The main aim of pervasive systems is to provide services invisibly by being aware of the surrounding context and retaining the ability for subsequent reaction. Such systems are characterized by their heterogeneity, observed in the diversity of software and hardware involved. They are also characterized by the abundance of small scattered devices, limited network facilities, and high mobility. As research in pervasive systems evolved, many useful applications emerged. As the need for ubiquitous development and deployment of such systems became stronger, relevant software engineering practices specific to such types of systems had to be developed. One of the big challenges herein includes the design of such systems, and in this paper we present an approach for generating architectures for pervasive systems based on the selection of features specified by system designers.

The rest of the paper is organized as follows: In section 2, we cite some related research work. In section 3, we explain how we categorized the architectural features that we collected by studying over fifty pervasive systems architectures. Section 4 explains how we automatically generate architectures by selecting a set of desired features, section 5 presents two case studies and finally section 6 concludes the paper.

2. Related Work

During our research, we came across different software product lines (SPL) that target distributed, embedded, adaptive, pervasive, and data-intensive systems. For example, a software product line that targets distributed and embedded systems is proposed by Scheidemann [4]. The SPL is designed for configuring a vehicle control system by exploiting the commonalities and variability in the requirements and the features that could be found in a car. The approach presented by Schmoelzer et. al. in [7] describes the application of component based and model driven development in building product lines for data-intensive systems. Data intensive systems handle data processing, visualization and storage, and are usually architected through the application of multiple tiers in their relevant architectures. In Hallsteinsen et. al's [6] work, product line techniques are used to build adaptive systems which adjust their properties and resources according to user needs and resource constraints at runtime. The approach presented by Cetina et. al [1], is based on Model Driven Development (MDD) and the variability modeling principle. It utilizes variability modeling and the available resources to get the most efficient reconfiguration of the software system to match users' goals. No significant effort has however been made to create reference architectures for pervasive systems with sufficient embodiment of the required commonality and variability amongst such types of systems.

Our study of a plethora of pervasive systems architectures led us to the exploration of commonalities and variability in such kinds of architectures. A set of architectural features common to these types of systems was specified and mapped to a given set of software components [3], which are subsequently aggregated to build an automatically generated architecture for a system comprised of a subset of the selected features.

3. Pervasive Systems Categorization

Our first attempt to automatically generate architectures was to study over fifty different architectures built for pervasive systems, and to capture the features inherent in such architectures. The outcome of the study we presented in [5] is visualized in Figure 1, and shows the various architectural features we found and subsequently categorized. We used the Feature Model Plug-in (FMP), developed by the Generative Software Development Lab at Waterloo University, and Eclipse to represent the categories [2] and inherent features. We found the natural interaction and sensor and actuator networks to be common features for all the pervasive system architectures that we have seen so far. However, the rest of the features, organized in thirteen different categories, are considered to be variables and could be selectively chosen according to the requirements put forth for new systems.

For each of the indicated features, we designed relevant architectures that could eventually be aggregated together based on a designer's choice for the incorporation of certain features required in a new architecture under development. Figure 2 and Figure 3 are samples of architectures we designed or extracted from the literature for the "natural interaction" and "sensor network" component features, respectively. We found these two components to be common to all pervasive systems that we have seen so far.

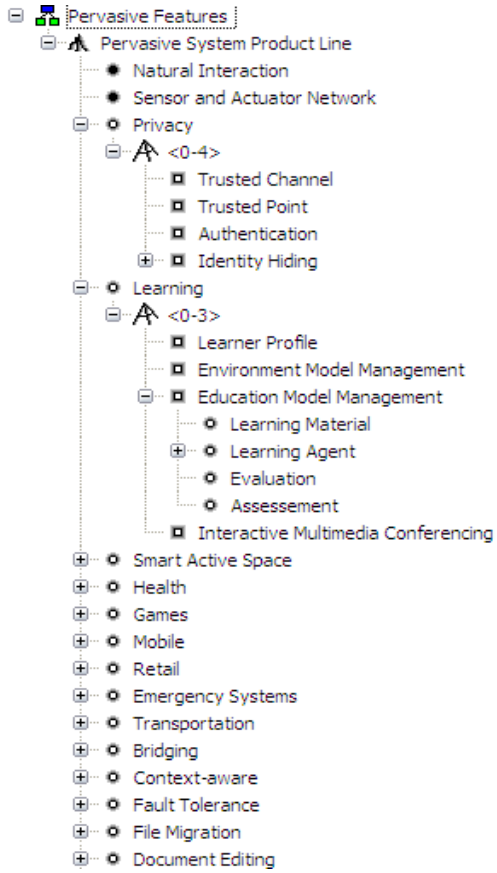


Figure 1: Pervasive Categorization using Eclipse and FMP

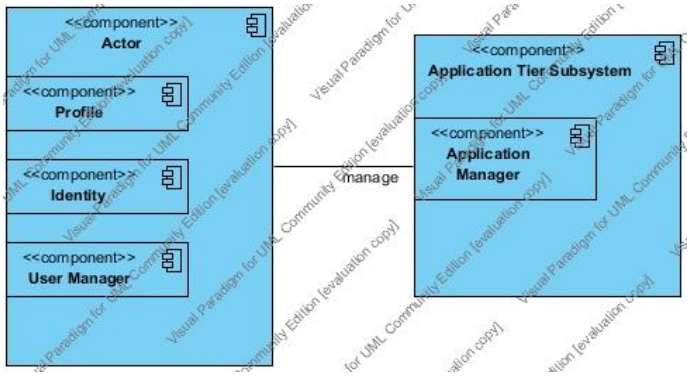


Figure 2: Natural Interaction Component Diagram

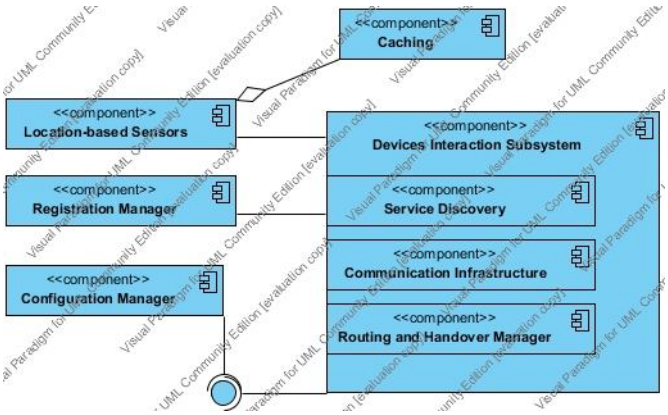


Figure 3: Sensor Network Component Diagram

4. The Architecture-Generation Process

We had to choose between creating a huge reference architecture that incorporates all the indicated architectural features with variability versus having smaller architectures and selectively aggregating their components together according to designer preference of selected features. We compared the two approaches and decided that it is more lucrative to adopt the latter approach. Having smaller architectures for the various features is better in our opinion for the following reasons:

1. *Expandability for new features and architecture scalability:* The generated architecture can be modified easily by adding new components from *different categories.*
2. *Easier to automate changes:* Applying or changing the connections among the components will be easier than in the bigger reference architecture, where it would be more difficult to trace and apply connections.
3. *Loosely coupled:* Architectures that are loosely coupled and can be replaced easily. The RAs for each category can be modified or replaced by another component diagram.

The process of generating architectures is shown in Figure 4. The needed features are first selected using the Feature Modeling plug-in (FMP) [2] with Eclipse by the system designer. Visual Paradigm for UML [7] was used to represent the component architectures. The selected features and the architecture diagrams are then exported in the form of XML documents. If any modification is done in the component architectures through Visual Paradigm after exportation, the XML document must be re-exported to reflect the updates. The reason behind using XML during the generation of the architectures is that XML is easier and better for standardizing the processing among the different tools used in our approach.

By using Visual Studio 2008, we developed a program in C# and Windows Forms that:

1. Maps the generated XML diagrams to the XML document converted from the features. This is done by mapping each feature to the corresponding component or set of components.
2. Adds these components to the generated component diagram.
3. Performs a second iteration over the generated component diagram in order to remove the unnecessary connections and to glue together the unconnected components that come from different categories according to a pre-defined lookup table.

The lookup table is manually pre-built before running the C# program. It is developed by collecting the matching components together from the different categories and checking if there could be any dependency among them. A dependency is identified if a component reads/writes/uses another one. In other words, if interactions are found by the designer between components, they are appended to the lookup table. For the lookup table structure, as shown in Figure 5, each line expresses a connection between 2 components by declaring the component names separated by a comma ‘,’. For example, the “Shopping Cart” component, which is used in retail systems, has a connection to the “Application Tier Subsystem”, which is a component of the actor. The “Shopping Cart” utilizes the “Application Tier Subsystem” by accessing the different retail applications that the actor is using. In other words, if the actor has a retail application tha he/she uses in setting preferences and compiling the shopping list, “The Application Tier subsystem” will act as the bridge between the “Actor” and the “Shopping Cart” with the correct wrappers to ensure they understand each other. The final generated XML document is readable through Visual Paradigm for UML and the component diagram can be viewed from there.

The generated architecture is still a first increment. More software engineering intervention is needed after generating the architecture to verify the architecture and make sure that it does not have blackouts or unnecessary components.

5. The Case Studies

We devised two scenarios for generating architectures. The first one is a combination of retail and context awareness while the other is a combination of retail and health.

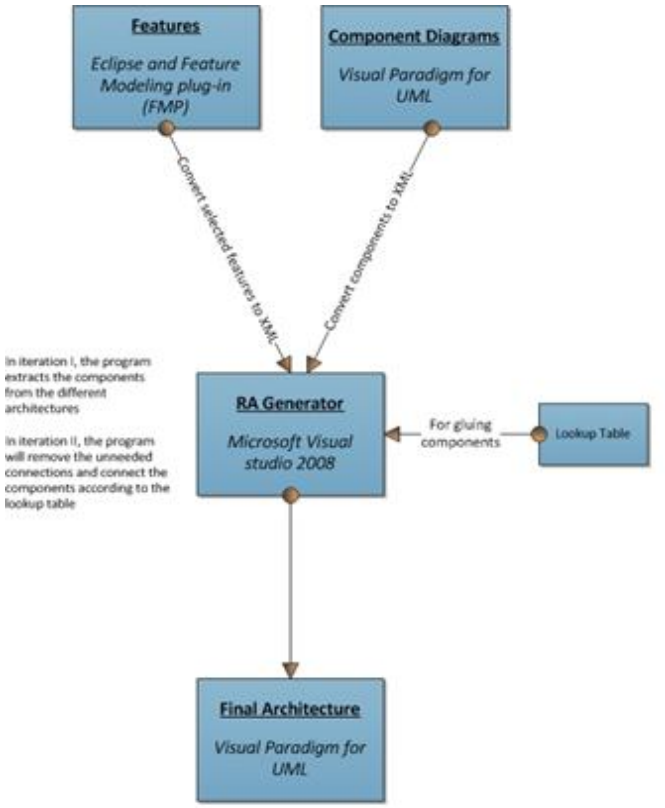


Figure 4: The Architecture Generation Process

```
Application Tier Subsystem,Shopping Cart
Application Tier Subsystem,Devices Interaction Subsystem
Application Tier Subsystem,Drug Manager
Actor,Health Sensors
Tracking Subsystem,Location-based Sensors
Mobile Manager,Transportation Management Tier
Application Tier Subsystem,Mobile Manager
```

Figure 5: Lookup table Sample

5.1 Retail and Context-Awareness

Scenario: While walking in a mall or a supermarket, one should be notified whether co-located entities have common interests. The system should detect if a person is shopping alone or with others, and accordingly, it notifies him/her with the common interests among the group.

According to the above scenario, we selected the features needed in order to generate the system as shown in Figure 6. The selected features are:

- Actors
- Sensors Network
- Shopping cart
  - Screen
  - Internet
- Bar Code Reader
- RFID Reader
- Transcoder
- Context Management
  - Model
  - Context Repository
  - Context Reasoner
- Situation Repository
- Preference Repository

Figure 6 shows how the features are selected from within our tool.

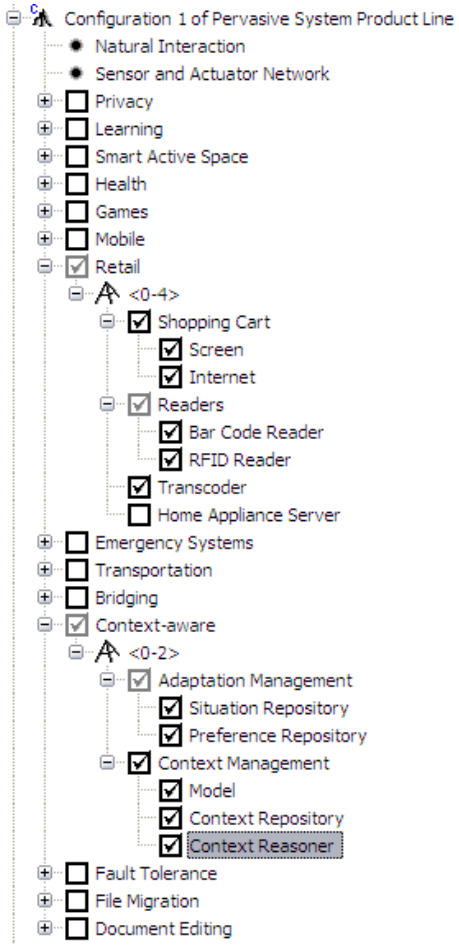


Figure 6: Configuration of Retail with Context-awareness

By applying the feature-driven approach and the architecture genration process described above, the resulting architecture is as shown in Figure 7. The subsystems included in the architecture are natural interaction, as well as sensor networks as described earlier in Figure 2 and Figure 3, respectively. Other subsystems are mapped from the selected features as shown in Figure 6. According to the lookup table, the "Application Tier Subsystem" is connected to the "Shopping Cart" and the "Application Tier Subsystem" is connected to the "Devices Interaction Subsystem". The "Application Tier Subsystem" is

considered a focal component that enables the system users to customize and personalize the system to their preferences.

5.2 Retail and Health

Scenario: While walking in a store or a mall; a need might arise to monitor a health condition. If a person is suddenly faced with a certain health condition, he/she should be notified with the nearest pharmacy, clinic or hospital according to the situation.

The needed features that we selected according to the discussed scenario are:

- Actors
- Sensors Network
- Shopping cart
  - Screen
  - Internet
- Bar Code Reader
- RFID Reader
- Transcoder
- Drug Management
- Health Monitoring Network

From the above scenario, the generated architecture is shown in Figure 8. The following categories were covered by the scenario: natural interaction: sensor network, context-awareness, retail, and health. The lookup table connected the "Application Tier Subsystem" to the "Drug Manager", and the "Actor" to the "Health Sensors". The "Drug Manager" component gets orders from the "Application Tier Subsystem" according to the customizations order. The "Actor" subsystem sends and receives data from the "Health Sensors" subsystem about the current situation of the person's health condition.

We selected the above two simple architectures to demonstrate the capability of the tool and the research done in generating architectures as well as to show that the architectures are loosely coupled. By changing some of the features, the final architecture and its functionality could totally change into a new architecture.

6. Conclusion

In this paper, we discussed our feature-driven approach for generating pervasive system architectures based on selected features that are desired in a particular system. The selected features are mapped to predetermined component diagrams which are glued together according to a lookup table to generate the desired architecture. We showed two case studies with their requirements, and indicated how our approach can be used to generate a pervasive architecture. The generated architecture must be subsequently verified through human intervention. Our work in progress includes refining a metric suite using the work in[9], [10], [11], [12], [13], [14], [15] and [16] to be used in evaluating the generated architecture in comparison to the architectures generated by human subjects for the same scenarios.

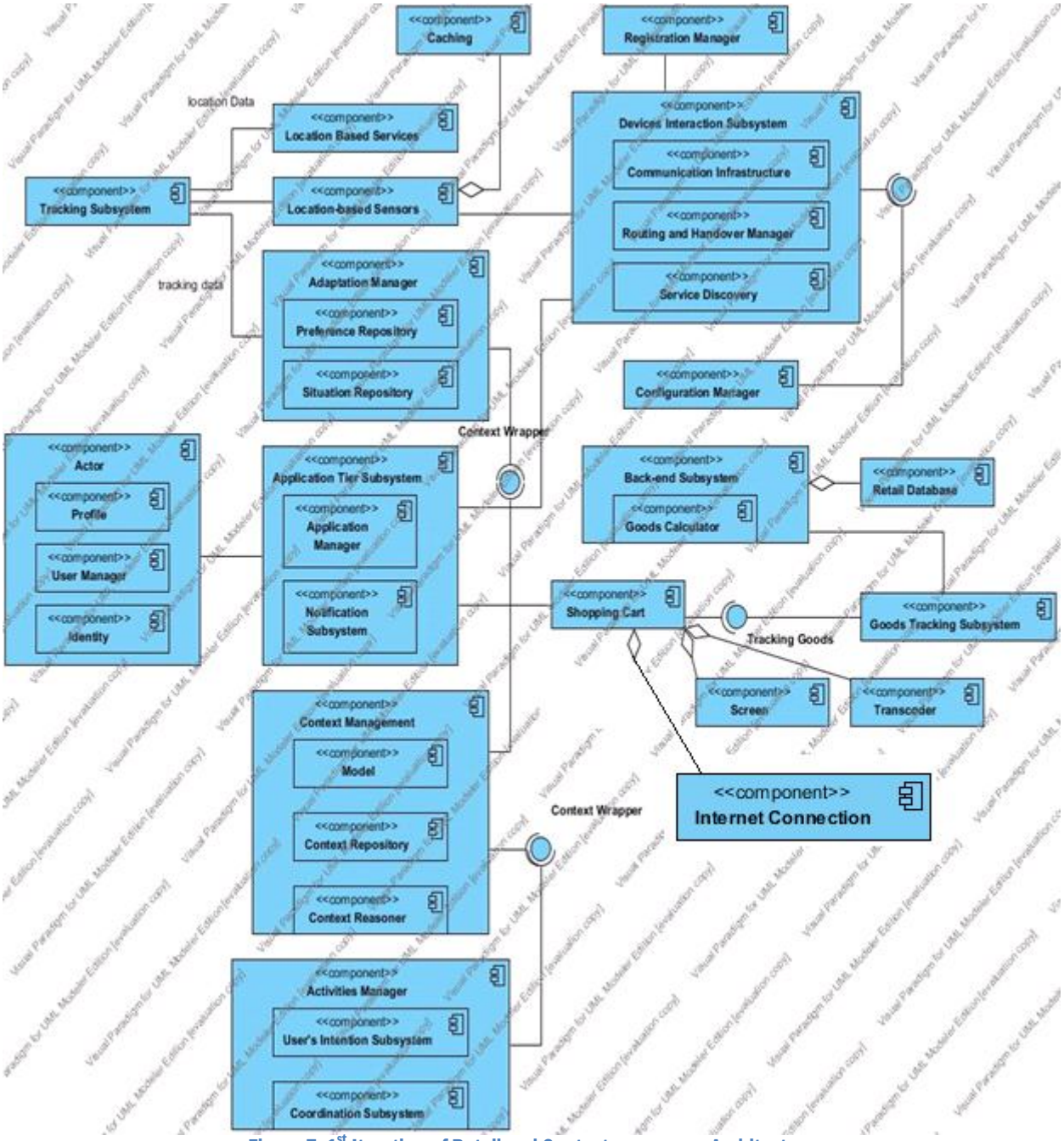


Figure 7: 1st Iteration of Retail and Context-awareness Architecture

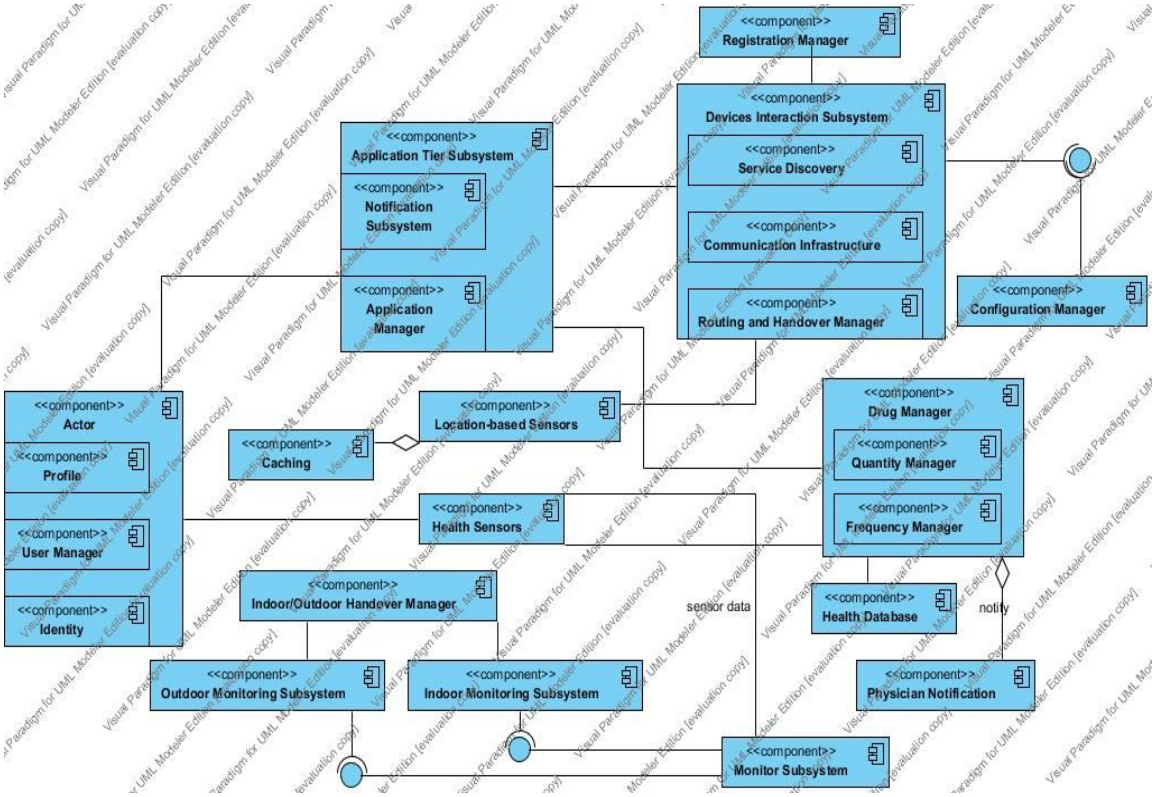


Figure 8: 1st Iteration for Retail and Health Architecture

REFERENCES

[1] Carlos Cetina, Joan Fons, Vicente Pelechano, "Applying Software Product Lines to Build Autonomic Pervasive Systems" 12th International Software Product Line Conference (SPLC 2008), pp. 117-126.

Feature Modeling Plug-in (FMP). An Eclipse plug-in for editing and configuring feature models. http://gsd.uwaterloo.ca/fmp

[3] George T. Heineman, William T. Councill, Component-based software engineering: putting the pieces together, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, 2001.

[4] Kathrin D. Scheidemann. "Optimizing the Selection of Representative Configurations in Verification of Evolving Product Lines of Distributed Embedded Systems." *10th International Software Product Line Conference (SPLC'06).* splc, 2006. Pages: 75-84.

[5] Mostafa Hamza, Sherif G. Aly. "A Study and Categorization of Pervasive Systems Architectures: Towards Specifying a Software Product Line". *In WorldComp 2010: World Congress in Computer Science, Computer Engineering, and Applied Computing*.

[6] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch. "Using Product Line Techniques to Build Adaptive Systems". *In SPLC'06: 10th Int. Software Product Line Conference*, pages 141–150, Washington, DC, USA, 2006. IEEE Computer Society.

[7] Schmoelzer, G., C. Kreiner and M. Thonhauser. "Platform Design for Software Product Lines of Data-intensive Systems." *Proceedings of the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications.* 2007. Pages: 109 - 120.

[8] Visual Paradigm for UML. http://www.visual-paradigm.com/product/vpuml/

SDMetrics. The Software Design Metrics tool for UML. http://www.sdmetrics.com/

[10] L. Briand, S. Morasca, V. Basili. "Property-based Software Engineering Measurement", IEEE Transactions on Software Engineering, 1996.

[11] M. Hitz and B. Montazeri, "Measuring Coupling and Cohesion In Object-Oriented Systems," Proc. Int'l Symp. Applied Corporate Computing (ISACC '95),Monterrey, Mexico, Oct.25-27, 1995.

[12] Edward B. Allen , Sampath Gottipati , Rajiv Govindarajan, Measuring size, complexity, and coupling of hypergraph abstractions of software: An information-theory approach, Software Quality Control, v.15 n.2, p.179-212, June 2007.

[13] Roger S. Pressman, "Software Engineering: A Practitioner's Approach", fifth edition. The McGraw-Hill Companies, Inc., New York

[14] "Can we measure architecture? " interview with Anja Fiegler. Enterprise and Solution Architect Certification & Resources. http://grahamberrisford.com/15%20Scale%20and%20Change/Can%20we%20measure%20architecture.htm

[15] Judith Barnard, A new reusability metric for object-oriented software, Software Quality Control, v.7 n.1, p.35-50, 1998.

[16] Xia Liu, Qing Wang, "Study on Application of a Quantitative Evaluation Approach for Software Architecture Adaptability," qsic, pp.265-272, Fifth International Conference on Quality Software (QSIC'05), 2005.

An Architectural Framework for the Improvement of the Ultra-Large-Scale Systems Interoperability

S. Shervin Ostadzadeh[1] **and Fereidoon Shams**[2]
[1]Computer Engineering Dept., Science and Research Branch, Islamic Azad University, Tehran, Iran
[2] Computer Engineering Dept., Shahid Beheshti University, Tehran, Iran

Abstract - *With a trend towards becoming more and more information based, enterprises constantly attempt to go beyond the accomplishments of each other by improving their information activities. This attempt depends on increasingly complex systems that will exceed the size of current systems and systems of systems by every measure. These systems are called Ultra-Large-Scale (ULS) systems. The sheer scale of ULS systems will change everything, necessitating that we broaden our understanding of software architectures and the ways we structure them. In this paper, we clarify the lack of an architectural framework for the ULS interoperability and suggest some early considerations towards proposing an architectural framework to improve the interoperability of ULS systems*.

Keywords: Ultra-Large-Scale Systems, Interoperability, Architectural Framework, Software Architecture, Software Engineering

1 Introduction

Software Engineering faces a lot of challenges nowadays, however, fundamental gaps in our current understanding of software and software development at the scale of Ultra-Large-Scale [27] Software-Intensive systems (ULSSIS) remains one important challenge that present serious obstacles in the technically and economically effective achievement of the Software Engineering goals. This is due to the fact that proper development of ULSSIS would have substantial impact on software engineering activities.

As systems grow larger and more complex, which eventually become ULSSIS, unprecedented demands on software architecture will emerge. The software architecture of a program or computing system is the structure or structures of the system, which comprises software elements, the externally visible properties of those elements, and the relationships among them [2]. In other words, software architecture characterizes the structure of a system. In general, architecture is the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [17].

According to ISO 15704 [16], an architecture represents a description of the basic arrangement and connectivity of parts of a system (either a physical or a conceptual object or entity), which is expected to create a comprehensive overview of the entire system when put together [8]. It should be noted that handling this large amount of information is quite challenging and needs a well-developed framework. This challenge will be intensified in ULS systems due to their large scale. Nowadays, various Information Systems Architecture (ISA) frameworks have been proposed; among them are Zachman Framework [28, 35], FEAF[1] [9], TEAF[2] [10], TOGAF[3] [23], E2AF[4] [26] and C4ISR [6]. Unfortunately, they do not provide all the required support for ULSSIS. The inability of current ISA frameworks to meet the demands of these systems, calls for breakthrough research in an ultra-large-scale architectural framework [27].

This paper presents an initial work on applying an architectural framework in ULS systems interoperability, which we believe has the potential to lead to the required breakthroughs.

The rest of the paper is organized as follows. In Section 2, we introduce some basic concepts and principles. Next, we present an interoperability model overview in section 3. We discuss our proposed approach in section 4. Finally, we make conclusions and provide some comments for the future work.

2 Basic concepts and definitions

In this section we briefly introduce some basic concepts and principles. We assume these remarks are necessary for the readers to clearly understand what we mean by the concepts used in this paper.

[1] Federal Enterprise Architecture Framework
[2] Treasury Enterprise Architecture Framework
[3] The Open Group Architecture Framework
[4] Extended Enterprise Architecture Framework

2.1 Ultra-Large-Scale systems

In 2006, SEI [27] published a report about some systems which were called as Ultra-Large-Scale systems. These systems will go far beyond the size of current systems and system of systems by every measure, such as, the number of the lines of code; the number of people employing the system for different purposes; amount of data stored, accessed, manipulated, and refined; the number of connections and interdependencies among software components; and the number of hardware elements.

There are some characteristics of ULS systems that will be revealed because of their scale [14, 27]: (1) decentralization; (2) inherently conflicting, unknowable, and diverse requirements; (3) continuous evolution and deployment; (4) heterogeneous, inconsistent, and changing elements; (5) erosion of the people/system boundary; (6) normal failures; (7) new paradigms for acquisition and policy. These characteristics undermine current, widely used, information systems framework and establish the basis for the technical challenges associated with ULS systems.

2.2 Interoperability

The definitions on interoperability have been reviewed in [30, 31]. Broadly speaking, interoperability has the meaning of coexistence, autonomy and federated environment, whereas integration refers more to the concepts of coordination, coherence and uniformization [8]. Regarding the degree of coupling, when the components are totally dependent on each other and inseparable, the system is called 'tightly coupled' or fully integrated. On the contrary, the components in a 'loosely coupled' system (such as ULS systems) are connected via a communication network that makes the interactions possible. The components in these systems can exchange data/services while preserving locally their own logic of operations. This is called interoperability.

Another point of view is given by ISO 14258 [32]. Two systems are considered as 'integrated' if there is a detailed standard format for all constituent components. 'Interoperability' is more related to the unified approach where there is a common meta-level structure across constituent models, providing a means for establishing semantic equivalence or the federated approach where models must dynamically accommodate rather than having a predetermined meta-model [8].

2.3 Enterprise architecture

According to ISO 15704 [16], an 'enterprise' is one or more organizations sharing a definite mission, goals and objectives to offer an output such as a product or a service. An enterprise consists of people, information, and technologies; performs business functions; has a defined organizational structure that is commonly distributed in multiple locations; responds to internal and external events; has a purpose for its activities; provides specific services and products to its customers [25].

According to ISO 15704 [16], an architecture is a description of the basic arrangement and connectivity of parts of a system (either a physical or a conceptual object or entity). Enterprise Architecture (EA) is a comprehensive view of an enterprise. EA shows the primary components of an enterprise and depicts how these components interact with or relate to each other. EA typically encompasses an overview of the entire information system in an enterprise; including the software, hardware, and information architectures. In this sense, EA is a meta-architecture. As regards, EA contains different views of an enterprise, including, work, function, process, and information, it is at the highest level in the architecture pyramid.

According to the IFAC–IFIP Task Force [33] and ISO 15704 [16] there are two types of architectures that deal with enterprise integration: (1) System architecture (sometimes referred to as 'Type 1' architecture) that deals with the design of a system; (2) Enterprise-reference architecture (sometimes referred to as 'Type 2' architecture) that deals with the organization of the development and the implementation of a project such as an enterprise integration or other enterprise development program. In other words, Type 1 architecture describes a system or sub-system regarding its structure and behavior. However, Type 2 architecture implies a framework that organizes necessary concepts and activities/tasks to design and build a system [8]. Zachman Framework falls into the Type 2 architecture.

2.4 Zachman framework

In 1987, an IBM researcher, named John A. Zachman, proposed a framework for Information System Architecture [35], which is now called Zachman Framework (ZF) [28]. A framework is a classification schema that defines a set of categories into which various things can be arranged. An enterprise architecture framework is a way of organizing and classifying the types of information that must be created and used for the enterprise architecture. Zachman states that "The Framework for Enterprise Architecture is a two dimensional classification scheme for descriptive representations of an Enterprise" [37]. ZF is a two dimensional information matrix consisting of 6 rows and 6 columns. Figure 1 depicts Zachman Framework.

Zachman Framework	Data	Function	Network	People	Time	Motivation
Planner						
Owner						
Designer						
Builder						
Contractor						
Functioning Enterprise						

Figure 1: The Zachman Framework

The rows describe the perspectives of various stakeholders. These rows starting from the top include: Planner (Scope); Owner (Enterprise Model); Designer (System Model); Builder (Technology Model); Contractor (Detail Representation); and Functioning Enterprise. The columns describe various abstractions that define each perspective. These abstractions are based on six questions that one usually asks when s/he wants to understand a thing. The columns include: Data (What is it made of?); Function (How does it work?); Network (Where are the elements?); People (Who does what work?); Time (When do things happen?); and Motivation (Why do things happen?). To find complete cell definitions of ZF refer to [36].

3 Interoperability models

Since the beginning of the last decade, most recent work on architecture development is focused on careful planning and improving an enterprise interoperability framework. Conventionally, such a framework is primarily concerned with establishing a mechanism to describe the concepts, the problem and the knowledge on enterprise interoperability in a more structured manner. This section will survey some recent interoperability models.

3.1 LISI reference model

The LISI [6] (Levels of Information Systems Interoperability) approach developed by C4ISR Architecture Working Group (AWG) during 1997, is a framework to provide the US Department of Defense (DoD) with a maturity model and a process for determining joint interoperability needs, assessing the ability of the information systems to meet those needs, and selecting pragmatic solutions and a transition path for achieving higher states of capability and interoperability. Figure 2 depicts the LISI Reference Model.

A critical element of interoperability assurance is a clear prescription of the common suite of requisite capabilities that must be inherent to all information systems that desire to interoperate at a selected level of sophistication [8]. Each level's prescription of capabilities

Description	*Computing Environment*	*Level*	P	A	I	D
Enterprise	Universal	4	Enterprise Level	Interactive	Multi-Dimensional Topologies	Enterprise Model
Domain	Integrated	3	Domain Level	Groupware	World-wide Network	Domain Model
Functional	Distributed	2	Program Level	Desktop Automation	Local Networks	Program Model
Connected	Peer-to-Peer	1	Local/Site Level	Standard System Drivers	Simple Connection	Local
Isolated	Manual	0	Access Control	N/A	Independant	Private

Figure 2: LISI Reference Model [6]

must cover all four enabling attributes of interoperability known as PAID, namely, Procedures, Applications, Infrastructure, and Data.

The LISI approach is focused on developing interoperability in US military sector. It is also used as a basis to elaborate other interoperability maturity models such as Organizational Maturity Model [34] and Enterprise Interoperability Maturity Model in ATHENA Integrated Project [1].

3.2 IDEAS interoperability model

The IDEAS Interoperability Framework (fig. 3) was developed by IDEAS project on the basis of ECMA/NIST Toaster Model, ISO 19101, ISO 19119 and was augmented through the quality attributes and intended to reflect the view that "interoperability is achieved on multiple levels: inter-enterprise coordination, business process integration, semantic application integration, syntactical application integration and physical integration" [15].

	Framework 1st Level	Framework 2nd Level	ONTOLOGY	QUALITY ATTRIBUTES					
			Semantics	Security	Scalability	Evolution	Performance	Availability	Portability
ENTERPR. MODEL	Business	Decisional Model							
		Business Model							
		Business Processes							
	Knowledge	Organisation Roles							
		Skills Competencies							
		Knowledge Assets							
ARCHITECT PLATFORM	Application	Solution Management							
		Workplace Interaction							
		Application Logic							
		Process Logic							
	Data	Product Data							
		Process Data							
		Knowledge Data							
		Commerce Data							
	Communication								

Figure 3: IDEAS Interoperability Framework [15].

In the business layer, all issues related to the organization and the management of an enterprise are addressed. The business model is the description of the commercial relationships between an enterprise and the way it offers products or services to the market. The knowledge layer is concerned with acquiring, structuring and representing the collective/personal knowledge of an enterprise. The ICT system layer is concerned with the ICT solutions that allow an enterprise to operate, make decisions and exchange information within and outside its boundaries. The semantic dimension cuts across the business, knowledge and ICT layers. Quality attributes are a supplementary dimension of the framework. The considered attributes are: (1) Security; (2) Scalability; (3) Portability; (4) Performance; (5) Availability and (6) Evolution.

3.3 ATHENA interoperability framework

The ATHENA Interoperability Framework (AIF) [1] is structured into three levels: (1) The 'Conceptual' level is used for identification of research requirements and integrates research results; (2) The 'Applicative' level is used for the transfer of knowledge regarding application of integration technologies; and (3) The 'Technical' level is used for technology testing based on profiles and integrates prototypes. Figure 4 depicts the AIF.

The ATHENA Interoperability Framework and the IDEAS Interoperability Framework are considered complementary [8]. At each level of AIF, one can use the IDEAS interoperability framework to structure interoperability issues into three layers (business, knowledge and ICT) and a semantic dimension.

3.4 Other relevant interoperability models

In the UK, the e-Government Unit7 (eGU), has based its technical guidance on the e-Government Interoperability Framework (e-GIF) [11]. e-GIF mandates sets of specifications and policies for any cross-agency collaboration and for e-government service delivery.

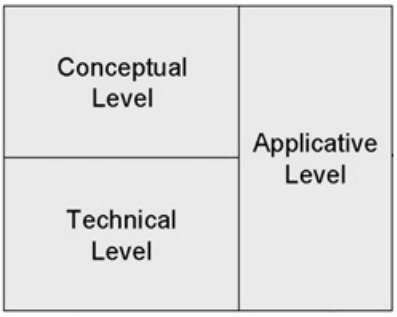


Figure 4: ATHENA Interoperability Framework

The European Interoperability Framework [12, 13] aims at supporting the European Union's strategy of providing user-centered e-Government services by defining as the overarching set of policies, standards and guidelines, which describe the way in which organizations have agreed, or should agree, to do business with each other.

The E-health interoperability framework [21] developed by NEHTA (National E-Health Transition Authority) initiatives in Australia, brings together organizational, information and technical aspects relating to the delivery of interoperability across health organizations.

The NATO C3 Interoperability Environment (NIE) [19] encompasses the standards, the products and the agreements adopted by the Alliance to ensure C3 interoperability. It serves as the basis for the development and evolution of C3 Systems.

Clark and Jones [34] proposed the Organizational Interoperability Maturity Model (OIM), which extends the LISI model into the more abstract layers of command and control support. Five levels of organizational maturity, describing the ability to interoperate, are defined. These include: (1) Independent; (2) Ad-hoc; (3) Collaborative; (4) Combined; and (5) Unified.

Layers of Coalition Interoperability (LCI) [29] is a framework to deal with possible measures of merit to be used to deal with the various layers of semantic interoperability in coalition operations.

The models previously discussed address a range of interoperability issues from technical to coalition organizational. SEI has developed the System of Systems Interoperability (SOSI) [18], which addresses technical interoperability (also covered by LISI, LCI, and NATO C3) and operational interoperability (also covered by OIM and LCI). However, SOSI goes a step further to address programmatic concerns between organizations building and maintaining interoperable systems. SOSI introduces three types of interoperability: (1) Programmatic, interoperability between different program offices. (2) Constructive, interoperability between the organizations that are responsible for the construction (and maintenance) of a system. (3) Operational, interoperability between the systems.

4 ULS interoperability framework

It is clear now that current approaches for defining, developing, deploying, operating, acquiring, and evolving ULS systems, as described in SEI report [27], will not suffice. We can consider ULS systems as cities or socio-

technical ecosystems, while our current knowledge and practices are geared toward creating individual buildings or species. This leads us to define a research discipline that is needed for new solutions.

In this section, we investigate the use of an architectural framework in order to improve ULS interoperability framework.

4.1 The problem

The scale of complexity and uncertainty in ULS system Design will be so great to resist treatments by traditional interoperability methods. According to SEI report [27], this is a new perspective: architecture is not purely a technical plan for producing a single system or closely related family of systems, but a structuring of the design spaces that a complex design process at an industrial scale will explore over time. Note that although breaking up an architecture into design spaces and striving for a set of coherent and effective design rules would seem to imply a significant degree of control of the overall design and production process, the design spaces, design rules, and the organizations will be continually adjusting and adapting to both internal and external forces.

The criticality of the research is justified by the fact that handling the large volume of information available in ULS systems is only feasible by utilizing a well-developed interoperability framework. A potential framework should broaden the traditional interoperability framework to include people and organizations; social, cognitive, and economic considerations; and design structures such as design rules and government policies.

The research should center on the development of an architectural framework to improve ultra-large-scale systems interoperability. We pose this question: *"Given the issues regarding the design of all levels of ultra-large-scale architectures, how can we organize and classify the types of information that must be created and used in order to improve ULS interoperability?"*

Current research shows that the problem stated in the previous paragraph is inherently broader and deeper than some of current successful ISA frameworks (such as Zachman Framework) [3, 4, 7, 20, 22].

4.2 Socio: the new concern

The SOSI can be consider as a significant initiative for ULS systems, however, as mentioned in SEI report [27], people will not just be users of a ULS system, rather, they will be part of its overall behavior. In addition, the boundary between the system and user/developer roles will blur. Just as people who maintain and modify a city, sometimes also live within the city, in a ULS system, sometimes a person will act in the role of a traditional user, sometimes in a supporting role as a maintainer of the system health, and sometimes as a change agent adding and repairing the functions of the system.

Assuming people to be part of the ULS system means that a new perspective has to be taken into account: Culture. Figure 5 depicts an extension to SOSI model to achieve ULS system socio-technical characteristics.

4.3 The proposed framework

Figure 6 depicts our proposed framework to improve ULS interoperability. As depicted, the framework applies Zachman Framework perspectives and abstractions in conjunction with the ULS interoperability model.

Zachman Framework is often referenced as a standard approach for expressing the basic elements of information system architecture, and is widely accepted as the main framework in ISA. In our work, we apply Zachman Framework as an initial start, and try to revise it by adding the required support for the special characteristics of the ultra-large-scale interoperability model. The proposed framework should be a spectrum of technologies and methods with software engineering, economics, human factors, cognitive psychology, sociology, systems engineering, and business policy.

The proposed framework uses three basic dimensions: (1) The abstract dimension is based on six general questions required to understand interoperability; (2) The perspective dimension is based on interoperability concerns in an enterprise; and (3) The final dimension is based on interoperability barriers in a socio-technical system of systems.

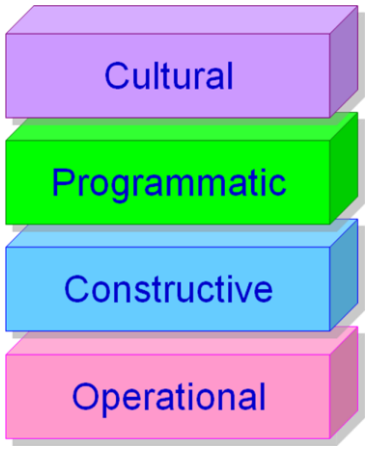


Figure 5: ULS Interoperability Model

The interoperability abstracts define the contents of interoperations:

- Data (What?): The interoperability of data handles information finding and sharing from heterogeneous data sources. These data sources possibly exist within different machines running different operating and data management systems.
- Function (How?): The interoperability of function takes care of identifying, composing and making various application functions work together.
- Network (Where?): The study of interconnecting the internal networks of companies is essential in a networked enterprise. This facilitates the creation of a common network for the whole enterprise. This type of interoperability focuses on the geometry or connectivity of the system's physical nodes.
- People (Who?): It focuses on the people and the manuals and the operating instructions or models they use to interoperate their tasks/duties.
- Time (When?): It is concerned with the life cycles, the timing and the schedules that are used to interoperate activities.
- Motivation (Why?): It focuses on goals, plans and rules that prescribe policies and ends which guide the enterprise interoperability.

The interoperability perspectives define various concerns of interoperation:

- Contextual: It describes the artifacts that provide the boundaries for the interoperability.
- Conceptual: It focuses on the artifacts that conceptually define the interoperability from the enterprise owners' perspective.
- Logical: It describes the artifacts that design the way interoperability will be realized systematically, quite independently of any technologies.
- Physical: It focuses on the artifacts that define the interoperability implementation based on the general technological constraints being employed.
- Out-of-Context: It describes the artifacts that specify the implementations for specific technological products being used for the interoperability.

The interoperability barriers address a range of interoperability issues from operational to cultural. In order to achieve Ultra-Large-Scale interoperation among systems, a set of cultural, management, constructive, and operational activities have to be implemented in a consistent manner. These activities require adding new and upgraded systems to a growing interoperability web:

- Operational issues define the activities within the executing system and between the executing system and its environment, including the interoperation with other systems.
- Constructive issues define the activities that develop or evolve system interoperability.
- Programmatic issues define the activities that manage the acquisition of system interoperability.
- Cultural issues define the activities that sustain the ULS system socio-technical characteristics.

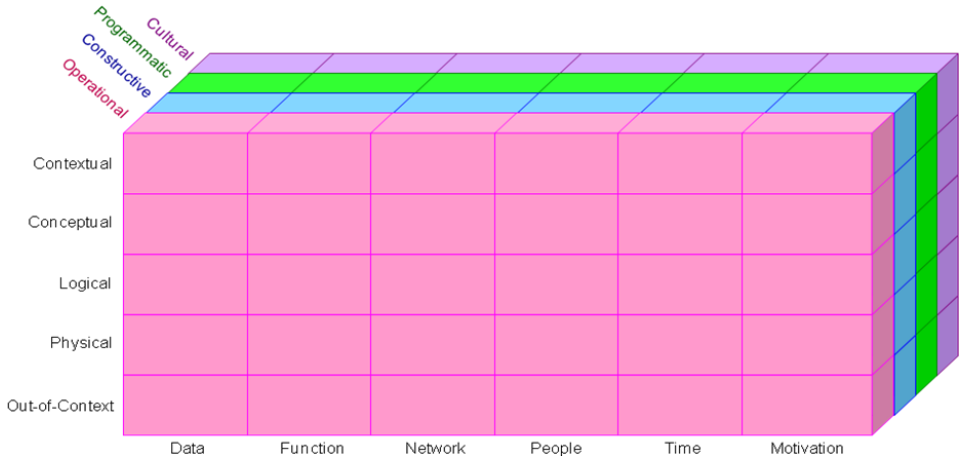


Figure 6: An ULS Interoperability Framework.

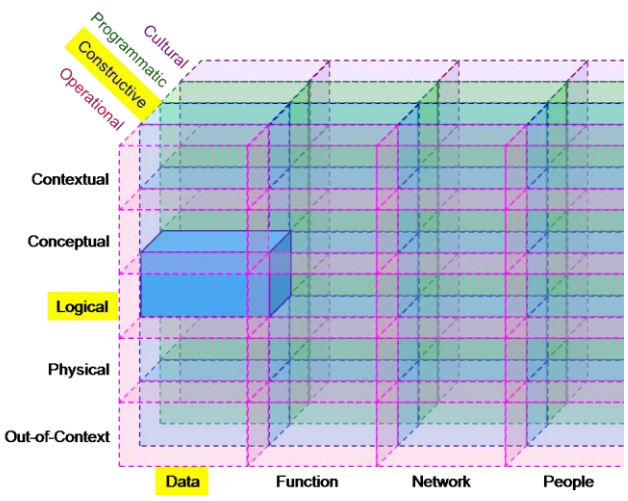


Figure 7: Data/Logical/Constructive Cell

Together, the abstract, perspective and barrier dimensions constitute the ULS interoperability framework. The two dimensional matrix (abstract × perspective) defines the contents of interoperations that take place in various levels of system perspectives. The third dimension enables to capture and to structure the type of interoperation. For example, Data/Logical/Constructive (fig. 7) represents the data that can be used for constructive interoperations in logical (system's designer) perspective.

5 Conclusions

In this paper, an architectural framework to improve ULS interoperability was proposed. The framework allows software architects to model various aspects of Ultra-Large-Scale systems interoperability. The proposed framework presents a classification schema for descriptive representation of a ULS system. The goal is that the framework be used to complement a full-structural model within the Ultra-Large-Scale interoperability. In particular, this approach will enable architects to: (1) design all levels of ULS architectures; (2) represent and analyze ULS interoperability; and (3) determine and manage ULS requirements. In the future work, one is expected to propose a methodology to help architectures model the framework cells.

6 References

[1] ATHENA. "Advanced Technologies for Interoperability of Heterogeneous Enterprise Networks and their Applications". FP6-2002-IST1, 2003.

[2] L. Bass, P. Clements, and R. Kazman. "Software Architecture in Practice". SEI Series in Software Architecture, Addison-Wesley Professional, 2004.

[3] S. Blanchette, et al. "U.S. Army Workshop on Exploring Enterprise, System of Systems, System, and Software Architectures". SEI, Carnegie Mellon University, USA, 2009.

[4] P. J. Boxer, and S. Garcia. "Enterprise Architecture for Complex System-of-Systems Contexts"; 3rd Annual IEEE International Systems Conference, Vancouver, Canada, 2009.

[5] Architecture Working Group (AWG). "C4ISR Architecture Framework, Version 2.0". USA Department of Defense (DoD), 1997.

[6] Architecture Working Group (AWG). "Levels of Information Systems Interoperability (LISI)". USA Department of Defense (DoD), 1998.

[7] P. Clements. "Exploring Enterprise, System of Systems, and System and Software Architectures". SEI Webinar, Carnegie Mellon University, USA, 2009.

[8] D. Chen, et al. "Architectures for enterprise integration and interoperability: Past, present and future"; Comput Industry (Ind), 2008.

[9] Chief Information Officers (CIO) Council. "Federal Enterprise Architecture Framework, Version 1.1". 1999.

[10] Department of the Treasury. "Treasury Enterprise Architecture Framework, Version 1". 2000.

[11] eGovernment Unit. "eGovernment Interoperability Framework (eGIF), version 6.1". 2005.

[12] EIF. "European Interoperability Framework". Brussels, 2004.

[13] EIF. "European Interoperability Framework for PAN-European eGovernment Services, Version 4.2". 2004.

[14] G. Goth. "Ultra-Large System: Redefining Software Engineering"; IEEE Software Journal, Vol. 25, Issue 3, 91—94, 2008.

[15] IDEAS. "IDEAS: Interoperability Development for Enterprise Application and Software Roadmaps". 2002.

[16] ISO 15704. "Industrial Automation Systems—Requirements for Enterprise-reference Architectures and Methodologies". 2000.

[17] IEEE Standards board. "Recommended Practice for Architectural Description of Software-Intensive Systems". IEEE-Std-1471-2000.

[18] E. Morris, et al. "System of Systems Interoperability". SEI, Carnegie Mellon University, USA, 2004.

[19] NC3A. "NATO C3 Technical Architecture Reference Model for Interoperability". NATO Consultation, Command, and Control Agency, 2003.

[20] NECSI. "Characteristics of Systems of Systems". NECSI: Complex Physical, Biological and Social Systems Project, 2004.

[21] NEHTA. "Towards an Interoperability Framework, Version 1.8". 2005.

[22] ODUSD. "Systems and Software Engineering: Systems Engineering Guide for Systems of Systems, Version 1.0". USA, 2008.

[23] Open Group. "The Open Group Architecture Framework (TOGAF), Version 9.0". USA, 2009.

[24] S. S. Ostadzadeh, et al. "A Method for Consistent Modeling of Zachman Framework"; Advances and Innovations in Systems, Computing Sciences and Software Engineering, Springer, 375—380, 2007.

[25] S. S. Ostadzadeh, et al. "An MDA-Based Generic Framework to Address Various Aspects of Enterprise Architecture"; Advances in Computer and Information Sciences and Engineering, Springer, 455—460, 2008.

[26] J. Schekkerman. "Extended Enterprise Architecture Framework Essentials Guide, Version 1.5". Institute For Enterprise Architecture Developments (IFEAD), 2006.

[27] SEI. "Ultra-Large-Scale Systems: Software Challenge of the Future". Carnegie Mellon University, USA, 2006.

[28] J. F. Sowa, and J. A. Zachman. "Extending and Formalizing the Framework for Information Systems Architecture"; IBM Systems Journal, Vol. 31, No. 3, 590—616, 1992.

[29] A. Tolk. "Beyond Technical Interoperability: Introducing a Reference Model for Measures of Merit for Coalition Interoperability"; Proc. 8th ICCRTS, USA, 2003.

[30] D. Chen, and F. Vernadat. "Enterprise interoperability: a standardisation view"; Kluwer Academic Publishers, 273—282, 2002.

[31] D. Chen, and F. Vernadat. "Standards on enterprise integration and engineering—a state of the art"; International Journal of Computer Integrated Manufacturing, 235—253, 2004.

[32] ISO 14258. "Industrial Automation Systems—Concepts and Rules for Enterprise Models". ISO TC184/SC5/WG1, 1999.

[33] IFAC–IFIP Task Force. "GERAM: Generalized Enterprise Reference Architecture and Methodology, Version 1.6.3". IFAC–IFIP Task Force on Architecture for Enterprise Integration, 1999.

[34] T. Clark, and R. Jones. "Organizational Interoperability Maturity Model for C2". Department of Defense, Canberra, Australia, 1999.

[35] J. A. Zachman. "A Framework for Information Systems Architecture"; IBM Systems Journal, Vol. 26, No. 3, 276—292, 1987.

[36] J. A. Zachman. "The Framework for Enterprise Architecture – Cell Definitions". ZIFA, 2003.

[37] J. A. Zachman. "The Zachman Framework: A Primer for Enterprise Engineering and Manufacturing". 2003.

Visualization Architecture for User Interaction with Dynamic Data Spaces in Multiple Pipelines

S. Prabhakar
6444 Silver Avenue, Apt 203, Burnaby, BC V5H 2Y4, Canada

Abstract – *Data intensive computational systems use multiple dynamically changing data sources. For the user to understand their behavior, visualization systems need to interactively set up new visualization scenarios from the existing visualizations. Designing such visualization systems require supporting complex interactions between diverse data sources and tools. The global nature of these interactions is a major impediment in the design and visualization time extension of such systems. We propose a principled approach that transforms the global interactions to fewer local interactions. The design is facilitated by a novel abstract software layer, which is based on multi-layered data flow architecture with multi-layered control. We also present a set of event based design patterns – sequences of local interactions among modules of software layer, which achieve the user interaction tasks. This software architecture is implemented in a Java based system called VisCoAdapt.*

Keywords*:* Architecture; Dataflow; Multi-layered control; Event based design patterns; Visualization systems; Human Computer Interaction

1 Introduction

An important class of Data Oriented Visualization Systems visualizes data from diverse and dynamically changing data spaces. Such data spaces are typically transformed by algorithms, which run independent of the visualization processes but incorporate guidance received from them. User interaction tasks require that visualization system support complex interactions between various tools and data sources. One such important task is composing an existing visualization scenario with new visualization scenarios, where each additional scenario supports interactions with data sources similar to the original scenario. Further, new interactions arise between various existing scenarios.

Design of such visualization systems is very complex as global interactions arise between various components. This is because various interactions arise between diversity of data sources and multiple tools. Further, the user driven visualization tasks require new type of control that manages interactions between components of visualization system from the display.

Dataflow architectures provide an important layer of abstraction for the integration of various software modules [9]. This is enabled by mapping various stages of visualization onto a single dataflow. While this is a powerful architecture, there are additional requirements on these visualization systems – incorporation of multiple diverse dynamic data sources, ability to support several user interactions and extensibility to multiple scenarios.

- During an interaction with visualizations, the user does not have explicit access to specific processes that transform data spaces or visualizations. The new architecture needs to deliver either the user input or the data source output to the appropriate destination. This requires deciding on the path and direction of dataflow within the architecture.

- The architecture should allow the user to create new scenarios from the existing ones. The new scenarios should maintain the same properties, as other scenarios, of interaction with the user, data sources and existing scenarios.

These requirements point to a critical issue in the design of visualization systems. The interactions between modules in the architecture are global in nature: each interaction can depend on the states of other modules in the architecture, some of them farther away in the path of dataflow. As a result, design of visualization systems becomes tedious, and extensibility to multiple scenarios during user interactions is unachievable. In order to address these requirements, we propose novel dataflow architecture. The central idea is to transform the global interactions between different modules into local interactions. This is achieved by the following novel aspects of architecture.

- The dataflow pipeline abstraction is at the level of abstract transformations over data. This abstraction unifies various kinds of transformations into a single layer. For example, visualization and data space transformations are specializations of this abstraction.

- Generally, in a dataflow architecture, each pipeline module has multiple functionalities. This is a major

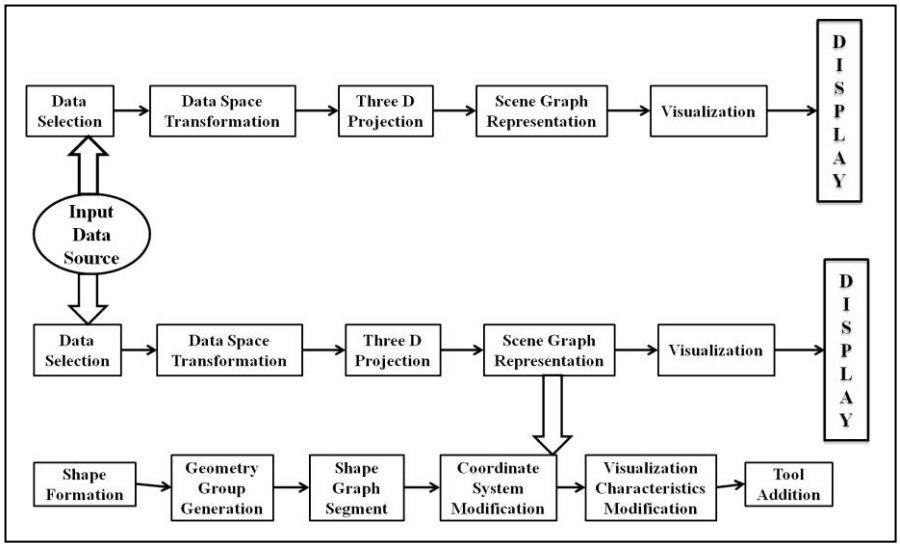


Figure 1. Dataflow Architecture with abstract data transformations as stages.

source for global interactions. In the proposed architecture, these functionalities are separated out into different control layers.

- The interactions between different visualization scenarios are enabled by interactions between pipelines, and are supported by new control layers. These include creation of new pipelines and interaction between pipelines.

- Modules with localized and isolated functionalities support local decisions for computing the data flow path. We propose event based design patterns that carry out the complex user interaction tasks as sequences of decision supported local event firings.

This architecture is completely implemented in Java in a system called *VisCoAdapt*. The visualized data is modeled using a scene graph, which is grounded in jReality [5].

2 Architecture

2.1 Pipeline Stages

In order to incorporate diverse data space and visualization transformations into a single dataflow, the *pipeline stages* are defined as abstract transformations. Figure 1 illustrates the architecture with two *pipelines* instantiated.

Each pipeline starts with a *data source* and ends with a *display*, and integrates multiple data space transformations into their stages. Each stage has the only function of transforming its input data into output data and is specified in terms of a set of constructors. Figure 1 illustrates three types of instantiations of data space transformations in pipeline stages. These transformations are used typically in many Machine Learning algorithms. Each pipeline includes three stages for Machine Learning: Data Selection, Data Space Transformation[1] and Three D Projection. The visualization stages transform the visualization properties of inputs to outputs. For example, the data points are organized into geometry groups. The data space transformation stages work with visualization stages to set up visualizations of various aspects of Machine Learning algorithms with which the user can interact. Each stage is associated with a set of parameters, which the user can modify by interacting with visualization.

Figure 1 also illustrates the hierarchical structure of pipeline: the Scene Graph Generation stage is further decomposed into various sub-stages. These sub-stages model the visualization data into a scene graph, which is displayed. Hierarchical decomposition masks some stages from the others in interaction. This makes the control of data flow and user interaction data manageable. This issue is discussed further in the section on Event based Design Patterns.

The functionalities of various stages in pipeline are briefly presented here. The *Data Selection* stage captures an important first step in Machine Learning algorithms – data selection from a given data set. The *Data Space Transformation* stage generates the target function learnt by using a Machine Learning algorithm [2]. The *Three D Projection* stage generates 3D projections of target functions by using algorithms such as Principal Component Analysis [4]. The output of this stage has geometry, such as a manifold [14], even though the input data to the pipeline may not be geometric. The *Scene Graph Generation* stage is central to user interactions as it provides the language, visualization and tools for representing and interacting with the data produced in earlier stages. As shown in figure 1, Scene Graph

[1] We use *data space transformation* to indicate a general transformation of data, and also as a name to a pipeline stage. Wherever there is a scope for confusion, we explicitly add *stage* in the latter case.

Generation stage contains six sub-stages. The *Visualization* stage has a set of displayers that display the scene graph. A variety of displayers, such as the one-panel or two-panel displayers, allow for various visualization configurations. Our displayers are grounded in the viewers of jReality [5].

The *Shape Formation* sub-stage uses Non-Uniform Rational B-Splines (NURBS) [9] to represent the data. This parameterizes visualizations for user interactions. The *Geometry Group* sub-stage organizes data is into groups of geometric objects for visualization and interaction purposes. The *Scene Graph Segment* sub-stage provides a language to represent the geometric objects and geometric groups along with coordinate system and tools for visualization. These scene graphs can be reconstructed based on the parameters received from visualization. We implemented our scene graph over jReality [5], a general scene graph for visualizations.

The *Coordinate System Embedding* sub-stage creates a coordinate system in which the geometry groups are embedded. The *Visualization Characteristics* sub-stage manages appearance parameters for the geometric objects and the coordinate system. The *User Interaction Tools* sub-stage creates and manages tools that enable the user to interact with visualizations. The tools allow the modifications and selections made to the visualizations to be parametrically communicated to the rest of the pipeline. Thus the tools act as another significant source of data to the pipeline.

2.2 Controlling Intra-Pipeline Communications

Each pipeline needs to support several functionalities – carrying out user interaction tasks, and responding to data input from preceding stages. The invocation and direction of data and control flows at each pipeline stage are conditional to the states present in the rest of the architecture.

It is not possible for a pipeline module to decide on the flow without knowing the states of other modules in the entire architecture. For example, a pipeline stage performs input data transformation if data becomes present either at stage input or at the corresponding stage in another pipeline, or the user performs a visualization interaction task relevant to the current stage. Another example of dependence of transformation on another event is the replication of a pipeline stage when the user chooses to add a new scenario or a preceding pipeline module is replicated.

In order to convert the non-local dependencies of data flow to local decisions, several control layers are added, three of which are shown in figure 2. The Controller layer decides the direction in which the data and control flow needs to be propagated. Each Adapter decides if the received request for data transformation is relevant to the current Stage Layer or not. The Factories layer creates a new pipeline from the current pipeline using modification data provided by the user. This layer is discussed in more detail in the next section. The control layers are designed such that they support a new kind of Event-based Design Patterns to implement data and control flows. These are discussed in a later section.

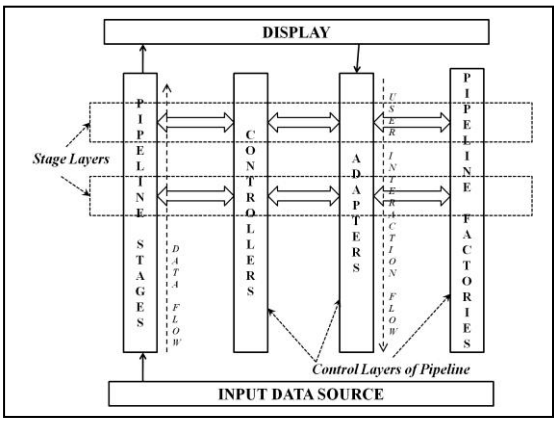


Figure 2. Multiple control layers of a pipeline.

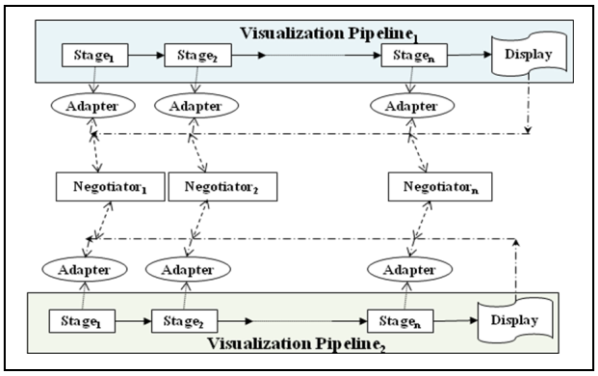


Figure 3. Communication between pipelines

2.3 Controlling Inter-Pipeline Communications

Each pipeline may be operating on a different set of visualization parameter values. For example, Data Selection algorithms in two different pipelines may be sampling data at different rates. But the pipelines need to coordinate for each user interaction task. This gives the user a comparative view of similar processes he/she is trying to understand. For example, the user may change the shape visualized in a scenario set up by a pipeline. These changes need to be translated to other pipelines.

This is done by the *Negotiator* layer. Each Negotiator module stores the parametric relationships between two stages in two different pipelines as shown in figure 3. Each Negotiator implements this relationship by forcing one of its stages to transform if its other stage is transformed due to a request coming from the rest of the architecture. Thus the transformation requests are localized in between two similar stages belonging to different pipelines within the context of visualization changes. In figure 3, the Adapter module fits the data received from the Negotiator to suit its pipeline stage.

3 Event-based Design Patterns

The architecture for visualization system needs to support complex interactions that arise due to (i) user's inputs at visualization, (ii) input data given to each pipeline, (iii) data inputs present at a stage, and (iv) negotiations that arise between multiple pipelines.

To change the global nature of these interactions to those local to *modules*[2] in the architecture, our solution has three aspects. Each pipeline is hierarchically organized, masking lower level interactions from other levels. The complex functionalities of stages are isolated and organized among multiple control layers. These two aspects have been discussed in section 2. We discuss the third aspect in this section - a set of *Event-based Design Patterns*, which are defined to enable each module to have decision-based local control of the paths for data and control flows within the hierarchy of stages and the control layers.

3.1 Design Patterns for User Interaction Tasks

An event-based design pattern is defined to carry out a user interaction task or data input. Each design pattern is a sequence of event invocations over modules within the architecture. Each module generates events locally based on the conditions present in the architecture. These conditions are made available to modules through the propagation of a set of properties. Each event type is associated with a set of properties. Based on the events received and the properties of those events, each module performs data flow tasks and fires a set of events to other modules in the architecture. Currently, VisCoAdapt contains several event based design patterns that cover various interaction tasks.

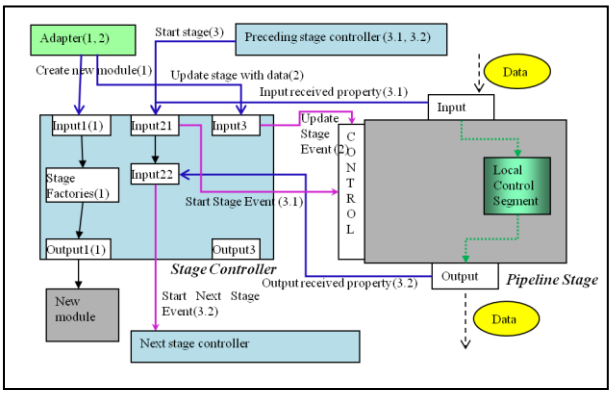


Figure 4. Event-based design pattern for the Controller module to control stage.

An example of an event-based design pattern captures the control of data transformations by the Controller within a stage. This pattern has a sequence of event invocations: (i) Controller module receives an event, (ii) the Controller module decides on the action to take based on the history of event invocations and property values, (iii) Controller module sends an event to the stage, (iv) the stage applies transformation on its input data, making the data available at the input of the succeeding stage, and finally (v) the Controller module sends an event to the succeeding Controller module. Figure 4 shows this event-based design pattern.

3.2 Design Patterns for Pipeline Creation

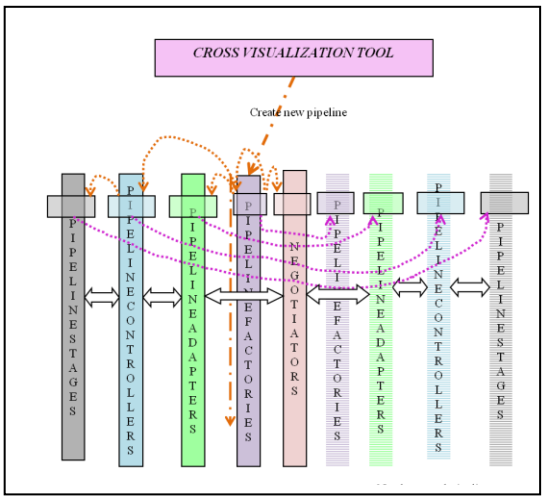


Figure 5. New pipeline creation by a sequence of events.

One of the user interaction tasks is to compose a complex visualization scenario by adding new scenarios to the existing scenarios. The user can invoke such a request using a user interaction tool at visualization. This request is handled by a control layer called *Pipeline Factories*, as shown in figures 2 and 5. This layer stores all the factories required to create a pipeline. It creates a new pipeline by sending a sequence of replicating events to all the modules of existing pipeline.

4 Experiments

This section presents a set of examples that illustrate the basic capabilities of the architecture of VisCoAdapt presented in section 2 and design patterns presented in section 3. The primary objective in all these experiments is to show that each user interaction based visualization task is realized through a sequence of event firings with localized decision making confined to modules of the architecture.

The experiments presented make some simplifying assumptions without loss of generality. The experiments start with a visualization system having one pipeline. All the data is presented to the pipeline from the input data source (figure 1). In these experiments, the input data source at the pipeline presents geometry rich data such as knot vectors, control points and degrees of the NURBS objects. The visualizations shown are at the display end of the pipeline (figure 1). The

[2] We reserve the term *module* to refer to any component of architecture. It can be a pipeline stage or it can be a component of a control layer.

Data Selection, Data Space Transformation, and Three D Projection stages perform identity transformations. This does not minimize the illustrative potential of the experiments as the specific functions used in data space transformations do not play any role in the abstraction required for the architecture.

4.1 Visualization without User Interactions

In this first experiment, the input data to the pipeline is the geometric information of a NURBS curve: a knot vector, control points, and the curve degree. Figure 6 shows the resulting visualization of the NURBS curve.

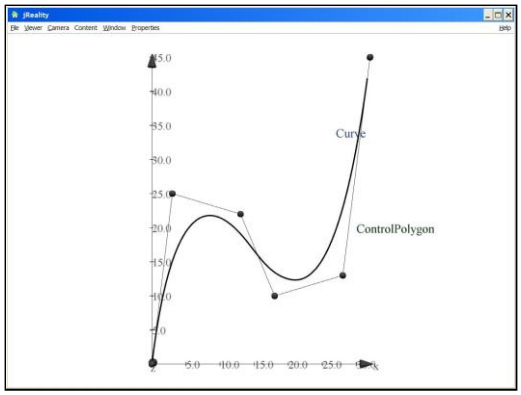


Figure 6. A NURBS Curve

The following is a sequence of steps of a design pattern that performs this visualization task.

1. Data becomes available at the input of the pipeline. This triggers an event and the Controller of Data Selection stage receives the event.
2. The Data Selection stage Controller fires an event requesting the Data Selection stage to apply stage transformation on its input data.
3. The Data Selection stage applies the transformation on input data. Since this stage has an identity transformation function, the input data is carried over to the output of the stage, without any changes. Since the output of Data Selection stage is connected to the input of Data Space Transformation stage, the data becomes present at its input.
4. The Data Selection stage Controller sends an event to Data Space Transformation stage Controller asking it to control its stage transformation.
5. The Data Space Transformation stage Controller receives the message that data is available at the input of its stage. It also receives an event from Data Selection stage Controller.
6. The steps 2 – 5 are repeated for each of the stages and sub-stages succeeding the Data Selection stage, producing the display shown in figure 6.

In our second example, the input data source to the pipeline is the geometric data of NURBS surface: U knot vector, V knot vector, Control Points, U degree and V degree. The design pattern consists of similar steps to those of NURBS curve presented earlier, and upon execution produces the visualization of NURBS surface as shown in figure 7.

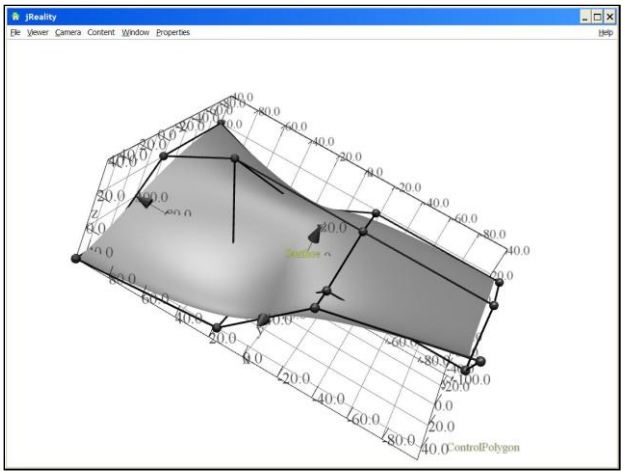


Figure 7. NURBS Surface and Control Patch.

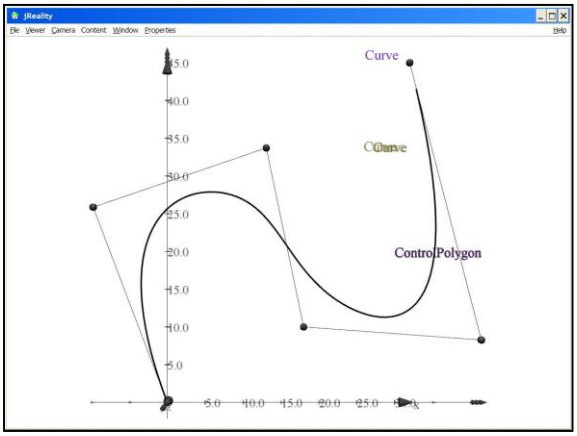


Figure 8. Shape modification of NURBS curve.

4.2 User Interaction for Shape Modification

Shape modification is a user interaction task, and figure 8 illustrates its effect on a NURBS curve (figure 6) by moving the control point. A new design pattern called Shape Modification Design Pattern produces the necessary event sequence. When the user drags a control point, this design pattern is triggered and it starts a new sequence of events inside the pipeline.

1. The Control Point Drag tool generates an event containing the context data. It is propagated along the Adapters layer (figure 2).
2. The Adapter of Shape Formation sub-stage recognizes it as relevant for its sub-stage. The Adapter triggers an event asking the Controller in the same Stage Layer (figure 2) to make necessary changes.

3. The Controller fires an event to the Shape Formation sub-stage asking it to apply the transformation on the data received from the tool.
4. The resulting data is made available at the input of the Geometry Group sub-stage.
5. Steps 2 – 5 of previous subsection are repeated for the succeeding sub-stages of Shape Formation sub-stage.
6. The resulting visualization shows the changes in the curve (shape) along with changes in the coordinate system.

Figure 9 shows the changes in the shape of a surface and its coordinate system, which are set up by the same sequence of steps as above.

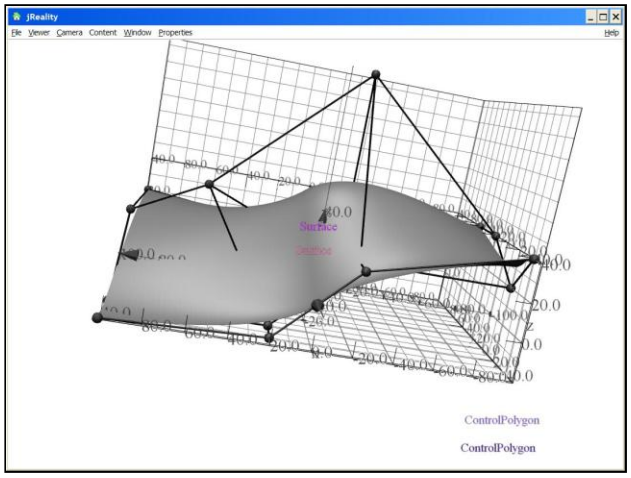


Figure 9. Shape modification of NURBS surface

4.3 User Interaction based Visualization Composition

The final example illustrates how the design pattern sets up a composed visualization, v_c from an existing visualization, v_1 and a new visualization, v_2 that meets some constraints. In this example, the user wants to add a visualization of a curve (v_2) that is 5 times larger than the curve in v_1 (figure 6). Each of $\{v_1, v_2\}$ in v_c has a unique pipeline and is able to provide similar kinds of user interactions with the pipeline as v_1. The added visualizations interact with the starting visualization to maintain a relationship the user intended. That is, the scaling between the curves in both the visualizations is maintained.

Figure 10 shows the composed visualization. We briefly summarize the steps that create the composition. These steps form the Visualization Composition Design Pattern. In this example, we use a simple implementation of a tool which allows a predefined set of constraints to be passed. Here, the constraints state that the new shape should obey a scaling factor of 5 with respect to the current shape.

1. The user selects the visualization creation tool and then sets up constraints. These constraints are passed to the Pipeline Factories layer (figure 5) of v_1.
2. This design pattern generates several events shown in figure 5, which replicate multiple layers of pipeline for v_1. This results in a new pipeline that can generate v_2. During this replication, the constraints are incorporated into the Negotiator in between the pipelines.

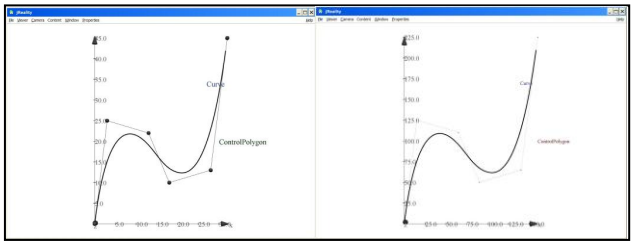


Figure 10. Composite visualization.

5 Related Research

Building software engineering architectures, to enable visualization of vast amounts of data, has been an active theme for research [8, 14, 16, 17, and 19]. In order to make the visualizations effective for each user, they need to be personalized so that the user has access to the patterns hidden in the vast amounts of data.

This personalization of visualizations is achieved in a number of ways. One of the approaches is to allow the user create pipelines by combining computational components in a dataflow [3, 7, 8, 12, 13]. This approach requires that the user understands the computational components, mechanism to interconnect them and a detailed knowledge of visualization. Another novel approach enables the user create visualizations by providing a comprehensive infrastructure that allows the application developer to explore collections of pipelines and combine them to create the applications [10]. It also captures detailed provenance of both application development and use [11]. VisCoAdapt shares these goals to provide a framework that the user can easily extend to build an application. It provides an infrastructure and the user builds an initial setup of application by extending a set of Java classes. The user is not required to have knowledge of the architecture. The user can build an application with the knowledge of algorithms and visualization attributes. For example, the user should be able to select a Machine Learning algorithm. The application can further be extended by the user at visualization time based on visualization requirements only.

Several attempts tried to integrate visualizations with other tasks such as problem solving [1, 8]. In VisCoAdapt, the integration is between Machine Learning algorithms and visualization. The algorithms are a part of pipeline, thus creation of new pipelines enables new interactions between modules of pipelines and the algorithms supported in those modules. The *negotiators* can be extended by the user thus capturing wide range of interactions between modules.

One of the requirements of a visualization system, which enables the user to have insights into data, is the support for

interaction between various visualizations [6]. In VisCoAdapt, we address this problem through pipeline interactions by using a mechanism for negotiations. The extensibility of the *negotiators* by the users supports wide range of interactions between pipelines.

6 Discussion and Conclusions

The data oriented visualization systems that visualize data from diverse and dynamically changing data spaces require a uniform mechanism to integrate diverse data sources. Further, the visualization systems are required to support a range of user interaction tasks. Design of such visualization systems faces a problem of global interactions between various components of the visualization system. To address this problem, we presented a new software architectural layer that has three features: a hierarchical dataflow architecture that integrates diverse data transformations and generations into stages of the pipelines; multiple layers of control to isolate and localize functionalities to modules; and several event based design patterns that implement complex user interaction tasks as sequences of localized decision-based event invocations.

This three part solution supports personalizing visualizations in dynamically emerging situations. The architecture breaks down the complex user interaction tasks into local tasks where specialized information provided by the user is applied. In personalization, the extensions provided by the user for Adapters and Negotiators play a significant role. For example, the Adapters decide the relevance of the given data to a Stage Layer. The Adapter can use simple rules or complex algorithms to make this decision. The Negotiators mediate between two pipelines in order to maintain the user specified relationship between the pipelines. An important consequence of the extensibility of Adapters and Negotiators is that the VisCoAdapt is highly scalable to various visualization tasks.

7 References

[1] Brodlie, K., Brankin, L., Poon, A., Banecki, G., Wright, H. and Gay, A. GRASPARC-A problem solving environment integrating computation and visualization. In Proceedings of IEEE Conference on Visualization (Oct 1993) 102-109.

[2] Hastie, T., Tibshirani, R., and Friedman, J. The Elements of Statistical Learning: Data Mining, Inference and Prediction. Springer, New York, 2001.

[3] IBM. OpenDX. http://www.research.ibm.com/dx.

[4] Joliffe, I.T. Principal Component Analysis. Springer, New York, 2002.

[5] jReality: http://www3.math.tu-berlin.de/jreality/

[6] Koop, D., Scheidegger, C.E., Callahan, S.P., Vo, H.T., Freire, J., and Silva C.T., VisComplete: Automating Suggestions for Visualization Pipelines. IEEE Transactions on Visualization and Computer Graphics, 14, 6 (2008) 1691-1698.

[7] Lee, E.A., and Parks, T.M., Dataflow Process Networks. Proceedings of the IEEE, 83, 5 (1995) 773–801.

[8] Macleod, R., Weinstein, D., de St. Germain, J., Brooks, D., Johnson, C., and Parker., S. SCIRun/BioPSE: Integrated problem solving environment for bioelectric field problems and visualization, in Proceedings of the Int. Symp. on Biomed. Imag.(Arlington VA, April 2004), 640–643.

[9] Piegl, L., and Tiller, W. The NURBS Book, 2nd Edition. Springer, New York, 1997.

[10] Santos, E., Lins, L., Ahrens, J., Freire, J., and Silva, C. VisMashup: Streamlining the Creation of Custom Visualization Applications. IEEE Trans. Vis. Comp. Graph 15, 6 (2009), 1539-1546.

[11] Silva, C.T., Freire, J., and Callahan, S.P. Provenance for Visualizations: Reproducibility and Beyond. Computing in Science and Engineering Journal 9, 5 (September 2007) 82-89.

[12] Silva, C.T., and Freire, J. Software Infrastructure for exploratory visualization and data analysis: Past, present, and future. Journal of Physics: Conference Series 125, 1 SciDac 2008 Conference (2008).

[13] The VisTrails Project. http://www.vistrails.org.

[14] Lee, J. A. and Verleysen, M. Nonlinear Dimensionality Reduction. Springer, New York, 2010.

Product Line Architectures for SOA

Mercy N. Njima[1], **Maurice H. ter Beek**[2], **and Stefania Gnesi**[2]
[1]IMT Institute for Advanced Studies, Lucca, Italy
[2]Istituto di Scienza e Tecnologie dell'Informazione, ISTI–CNR, Pisa, Italy

Abstract—*Service-oriented applications (SOA) are a standard-based and technology independent distributed computing paradigm for discovering, binding and assembling loosely-coupled software services. Software product lines (SPL) on the other hand allow a generic architecture to be configured and deployed in different instances. Product lines facilitate systematic reuse through managing variability. Product line engineering is a more established discipline and so may have more solutions to offer SOA and SPL. Thus, in this paper, we will look the synergies accruing from this powerful combination. We will then look at how the technique can be used to evaluate the latest smart energy management innovation in delivering return on investment for utilities.*

Keywords: Service Orientation, Product Lines, Smart Grid

1. Introduction

Product line technology, is increasingly finding its way to the software sector allowing companies to sustain growth and achieve market success [2]. The combination of Software Product Line (SPL) and Service-Oriented Application (SOA) development practices is becoming a new development paradigm that helps provide the answers to the need for agility, versatility and economies. SOA and SPL approaches to software development share a common goal. They both encourage an organization to reuse existing assets and capabilities rather than repeatedly redeveloping them for new systems. These approaches enable organizations to capitalize on reuse to achieve desired benefits such as productivity gains, decreased development costs, improved time to market, higher reliability, and competitive advantage [3]. Service Product Lines will achieve flexibility of network based systems through service orientation, but still manage product variations through product line engineering techniques [5]. Service features are selected and/or parameterized at runtime by a user or by a product itself when a certain contextual change or a new service provider is recognized. Service Orientation (SO) accommodates rapidly changing user needs and expectations. Adopting SO in practice for real software and system development has uncovered several challenging issues, such as:

- how to identify services,
- determining configurations of services that are relevant to users',
- current context, and
- maintaining system integrity after configuration changes.

Given that product line engineering approaches have been in the market for a longer period in the manufacturing industry they can be used to address these issues.

The paper is structured as follows. In Section 2, we review some work done to bring together software product lines and service oriented architectures. In Section 3 we present a successful approach to also apply the synergy to a real world problem. Section 4 introduces our case study in which we apply such techniques whilst Section 5, showcases variability concepts in the case study. We conclude in Section 6 with some remarks on future work.

2. Related work

There has been efforts to introduce the concepts of software product lines and service orientation: [2] explores their parallels and shows the applicability of software product line methods to service orientation. It also discusses the main obstacles to realizing the synergy between these cutting-edge technologies.

In [7], the authors outline the objectives of a workshop organized to explore how service-oriented architectures and software product lines can benefit from each other. Specifically they outline questions on

- how service-oriented systems can benefit from software product lines' variation management approaches to identify and design services targeted to multiple service-oriented systems and

- how software product lines can benefit from service-oriented architectures by employing services as a mechanism for variation within a product line.

The output from this workshop covers the area very well but it is beyond the scope of this paper to delve further.

There's also been work done in the application of these concepts to diverse domains from which we were motivated. Shokry et al. [12] apply the notion of dynamic runtime variability of software product lines in the embedded automotive software systems to create adaptable service-oriented architectures while [10] proposes an approach for service-oriented product line architectures that combines SPL and SOA concepts and techniques to achieve high customization and systematic planned reuse. They apply the techniques to conference management domain case study shown Figure 1.

The case study consists of a service-oriented product line that intends to produce customized service-oriented applications for the management of different conferences. Section 3 explores this work further in order to form a basis for our own case study.

3. Approach

The motivation for Medeiros et al. [10] stems from the fact that SOA lacks support for high customization and systematic planned reuse. In other words, despite the natural way of achieving customization in service-oriented applications, changing service order or even the participants of service compositions, services are not designed with variability to be highly customizable and reusable in specific contexts. Moreover, service artifacts, e.g., specifications and models, are not designed with variability as well. Hence, these artifacts cannot be easily reused by a family of service-oriented applications. Thus, SPL engineering, which has the principles of variability, customization and systematic planned reuse in its heart, can be used to aid SOA to achieve these benefits. In this way, service-oriented applications that support a particular set of business processes can be developed as SPLs. The aim is to achieve desired benefits such as productivity gains, decreased development costs and effort, improved time to market, applications customized to specific customers or market segment needs, and competitive advantage.

The approach for service-oriented product line architectures starts with an identification phase. It receives the feature model and the business process models as mandatory inputs, and produces a list of possible components, service candidates and service orchestration candidates for the product line architecture. Thus, these architectural elements can be reused in all products of the line. This phase is separated in component identification and service identification activities. Subsequently, there is a variability analysis activity. It receives the list of components and services identified previously, and defines and documents key architectural decisions regarding variability. In this activity, it is defined how the variability will be implemented within the services and components. Architecture specification activity concludes the approach. In this activity, the architecture is documented using different views in order to represent the concerns of the different stakeholders involved in the project. The approach takes the following steps:

- Component Identification: the components of the service-oriented product line are identified. A software component is a self-contained artifact with well-defined interfaces and is subject to third-party compositions. This activity starts with an analysis of the feature model to identify architectural component candidates. The purpose of this activity is to put features into modules (components) in order to design an architecture where components can be added or removed to generate customized products. Each of the modules identified in this activity will be an architectural component candidate for the service-oriented product line architecture.
- Service Identification: a set of service and service orchestration candidates that support the business processes are identified. Thus, as the services are supposed to support the business processes, it is reasonable to identify them from the business process models.
- Variability Analysis: variability is the ability to change or customize software systems. Improving variability in a system implies making it easier to do certain kinds of customizations. In addition, it is possible to anticipate some types of variability and construct a system in such a way that it is prepared for inserting predetermined changes. During the variability analysis activity, essential

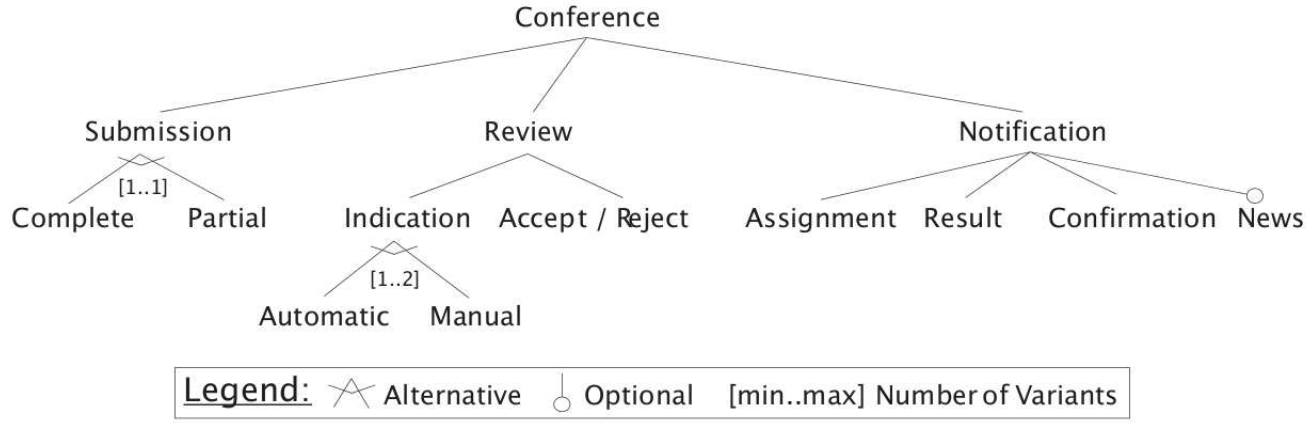


Fig. 1: Feature Model for the Conference Management Domain

architectural decisions about how the variability presented in the feature model and business processes will be implemented within services and components are defined.
- Architecture Specification: in this activity, the models and specification are produced with variability as all the artifacts of core assets development.

The model contains the following features:

- Submission: authors can submit their complete papers or, first submit the abstract, followed by the complete version. Complete and partial submissions are alternative features.
- Review: the indication of papers to reviewers can be made automatically and/or manually. Reviewers can also accept or reject paper indications. Automatic and manual indications are not exclusive, they can work together.
- Notification: the system can send information to reviewers about paper assignments. It can send acceptance or rejection (result) information to authors. It can also send event news, e.g., deadlines, and confirmation messages, e.g., paper or review submitted, to authors and reviewers. Event news notification is an optional feature. Assignments, confirmation and result notifications are mandatory.

4. Motivating Problem: The Smart Grid

Most of the world's electricity system was built when primary energy was relatively inexpensive. Grid reliability was mainly ensured by having excess capacity in the system, with unidirectional electricity flow to consumers from centrally dispatched power plants. Investments in the electric system were made to meet increasing demand – not to change fundamentally the way the system works. While innovation and technology have dramatically transformed other industrial sectors, the electric system, for the most part, has continued to operate the same way for decades. This lack of investment, combined with an asset life of forty or more years, has resulted in an inefficient and increasingly unstable system [4].

However, climate change, rising fuel costs, outdated grid infrastructure, and new power-generation technologies have changed the mindset of all stakeholders:

- Electric power causes approximately 25% of global greenhouse gas emissions, and electris utility companies are rethinking what the electricity system of the future should look like.
- Renewable and distributed power generation will play a more prominent role in reducing greenhouse gas emissions.
- Demand-side management promises to improve energy efficiency and reduce overall electricity consumption.
- Real-time monitoring of grid performance will

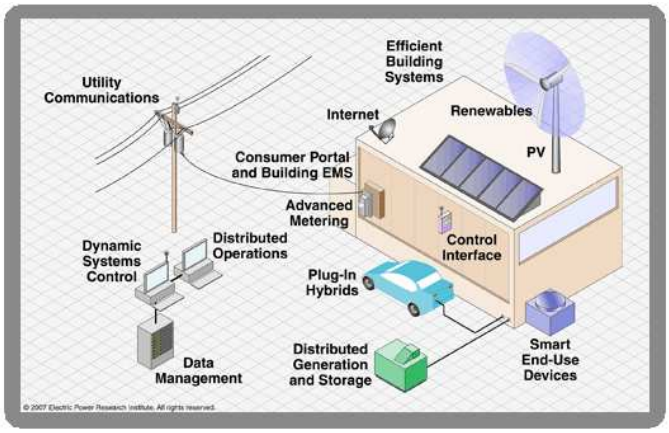


Fig. 2: Components of the smart Grid. Adopted from smartgrid.epri.com

improve grid reliability and utilization, reduce blackouts, and increase financial returns on investments in the grid.

These changes on both the demand and supply side require a new, more intelligent system that can manage the increasingly complex electric grid, see Figure 2. The energy community is starting to marry information and communications technology (ICT) with electricity infrastructure to enable the electric system to become 'smart'. Thus, the Smart Grid is the next-generation, managed electrical power system that leverages increased use of communications and information technology in the generation, delivery, and consumption of electrical energy. It consists of solutions based on both current and future telecommunication technologies for command and control, metering, and charging [9].

Electric utilities, in a reactive or proactive answer to these new challenges, are adding more intelligence and complexity in their distribution networks. As the grid becomes more intelligent and more complex, see Figure 3, the tools to operate it become increasingly important. Hence the need for interoperability from SOA and flexibility and variability from SPL [11]. The end result is then electricity provision as a service and the Smart Grid as a service product line.

In addition, it is essential to utilize product line technology to manage energy from the consumers point of view. This can be achieved by gathering metrics that report:

- Energy usage by product family;
- Real-time power draw by product;
- Energy consumption by device.

Services can then be used to achieve energy efficiency by helping customers virtualize infrastructure and perform energy efficiency benchmarking to effectively measure, monitor, and fine-tune energy usage. Sharing detailed analysis, recommendations, and design expertise services can also help energy management staff better understand, optimize, and control power to achieve significant cost savings [1].

5. Variability Modeling of Smart Grids

We introduce a feature-based approach to product line engineering and illustrate how this approach addresses the technical issues discussed relating to the smart grid. In this approach, a feature model (which captures the commonality and variability information of a product line) and feature-binding information about which features are included in products and delivered to customers, and when, are used as primary input to production plan development [8]. In product line engineering, a feature model plays a central role in the management and configuration of multiple products. Therefore, core assets should be identified based on features.

Feature Diagrams are a family of popular modeling languages used for engineering requirements in SPL represented as the nodes of a tree, with the product family being the root and have the following features:

- optional features may be present in a product only if their parent is present;
- mandatory features are present in a product if and only if their parent is present;
- alternative features are a set of features among which one and only one is present in a product if their parent is present [6].

We model the generic Smart Grid architecture as a family with basic components for basic products and specialized properties for some of the products such as:

- storage: will typically interconnect to the grid as a whole rather than being tied to balancing the output from a specific source;
- renewables: varies with weather, time, season and other intermittent effects. Since the grid is built to handle a lot of demand volatility and uncertainty we have to understand the impact of variable generation on grid reliability;

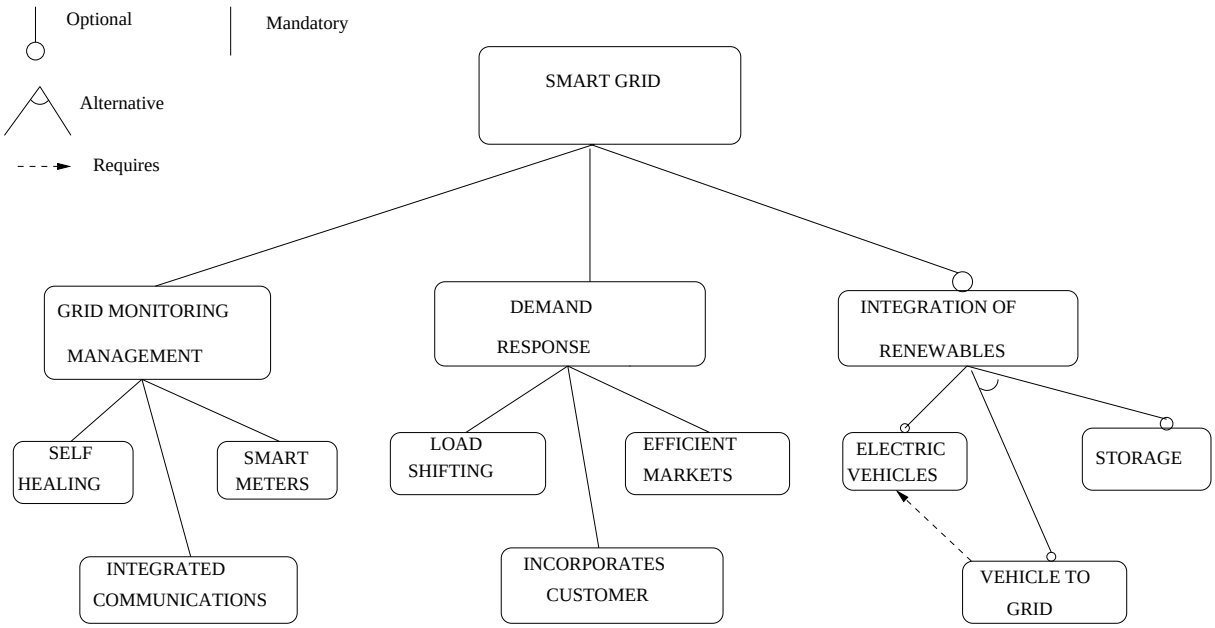


Fig. 3: Feature Model for the Smart Grid Family

- load shifting: practice of managing electricity supply and demand so that peak energy use is shifted to off-peak periods;
- vehicle to grid: establishing a viable transparent business model, guaranteeing the availability and controllability of electric vehicles and vehicle to grid (V2G) capacity, accurate forecasting of renewable energy supply and demand.

Load shifting and V2G can reduce the energy storage capacity required to maintain power quality.

Figure 4 shows the feature model for the smart grid family from which we can derive many other products depending on the needs of the consumer. We have the additional constraint 'requires' added to a feature diagram which is a unidirectional relation between two features indicating that the presence of one feature requires the presence of the other. We observe that features increment a product's functionality and more specifically that:

- the requires constraint obligates feature electric vehicles to be present whenever vehicle to grid (V2G) is.

Unfortunately, as the size of the model grows it becomes more difficult to derive products and deduce their behaviour. Furthermore, we see that most of the system behaviour modeled is static.

To counter this deficiency, we take advantage of the flexibility that comes with service orientation by introducing independently configurable service features [8]. A service feature represents a major functionality of the system that may be added or removed as a unit. From a SOA perspective, it can be viewed as an orchestrator of services and in this case products, in order to derive as many products as possible by initiating them as services. The service features in our smart grid model are grid monitoring management, demand response and integration of renewables all of which are independently configurable.

6. Conclusion and Future work

The realization of product line architectures for service oriented architectures is a very challenging issue for designing and developing systems required in the business environment. In this paper, we have pointed out examples from the automotive domain, conference management and built an energy management system from a static behavioural perspective. We have proposed that a service oriented approach can be exploited for such to support the flexibility of features.

We found that our approach would provide electric utility companies with an explicit way to develop their electricity provision products, organize their assets and take care of essential communication with customers. They would also be able to build specific products for

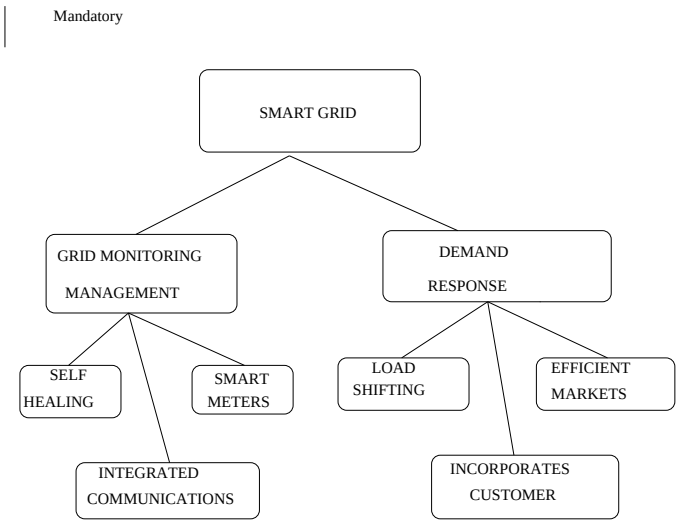


Fig. 4: Feature Model for a product without integration of renewables

different customers using an already existing architecture and thus, save time and bring such products to market quickly and efficiently.

Future work in this line of research will include identifying suitable models for describing feature oriented analysis in order to manage dynamic behaviour. Such dynamic requirements may come about when a consumer requests real-time assistance in managing their electricity load. From the utility's point of view, there maybe an oversupply of energy during off peak hours and they may need to take advantage of pre-existing service agreements with their consumers to utilise more. In addition, we are interested in working on storage using electric vehicles and the vehicle to grid feature given the complexity that comes from the requirement for the system to evolve adaptively according to changing needs.

References

[1] R. Aldrich and G. Mellinge. Cisco energy management: A case study in implementing energy as a service. `http://www.cisco.com/en/US/prod/collateral/switches/ps5718/ps10195/CiscoEMSWhitePaper.pdf`.

[2] O. Bubak and H. Gomaa. Applying software product line concepts in service orientation. *International Journal of Intelligent Information and Database Systems 2008*, 2(4):383–396, 2008.

[3] S.G. Cohen and R.W. Krut, editors. *Proceedings of the 1st Workshop on Service-Oriented Architectures and Software Product Lines: What is the Connection? (SOAPL'07)*, Technical Report CMU/SEI-2008-SR-006, Carnegie Mellon University, 2008.

[4] C. Feisst, D. Schlesinger, and W. Frye. Smart grid: The role of electricity infrastructure in reducing greenhouse gas emissions. `http://www.cisco.com/web/about/ac79/docs/wp/Utility_Smart_Grid_WP_REV1031_FINAL.pdf`.

[5] M. Galster. Describing variability in service-oriented software product lines. In *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, ECSA '10, pages 344–350, New York, NY, USA, 2010. ACM.

[6] K. Kang, S. Choen, J. Hess, W. Novak, and S. Peterson. Feature oriented domain analysis (FODA) Feasibility Study. Technical Report SEI-90-TR-21, Carnegie Mellon University, 1990.

[7] R. W. Krut and S. G. Cohen. Service-oriented architectures and software product lines: enhancing variation. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 301–302, Pittsburgh, PA, USA, 2009. Carnegie Mellon University.

[8] J. Lee, K. C. Kang, and S. Kim. A feature-based approach to product line production planning. In Robert Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 137–140. Springer Berlin / Heidelberg, 2004.

[9] E.M. Lightner and S.E. Widergren. An orderly transition to a transformed electricity system. *IEEE Transactions on Smart Grid*, 1(1):3 –10, jun. 2010.

[10] F. M. Medeiros, E. S. d. Almeida, and S. R. d. L. Meira. Towards an approach for service-oriented product line architectures. In *Proceedings of the Workshop on Service-oriented Architectures and Software Product Lines, San Francisco, CA*, 2009.

[11] D. Savio, L. Karlik, and S. Karnouskos. Predicting energy measurements of service-enabled devices in the future smartgrid. *International Conference on Computer Modeling and Simulation*, pages 450–455, 2010.

[12] H. Shokry and M. A. Babar. Dynamic software product line architectures using service-based computing for automotive systems. In *SPLC (2)*, pages 53–58, 2008.

[13] S. Trujillo, S. Apel, and Kaestner C. Product Lines that supply other Product Lines: A Service-Oriented Approach. In *Workshop on Service Oriented Architectures and Product Lines - What is the Connection? (SOAPL 2007) at the 11th International Software Product Line Conference (SPLC 2007), Kyoto, Japan, Sep*, 2007.

On Choosing Program Refactoring and Slicing Re-engineering Practice Towards Software Quality

Obeten O. Ekabua and Bassey E. Isong
Department of Computer Science and Information Systems,
University of Venda, Private Bag X5050,
Thohoyandou 0950, South Africa.
(obeten.ekabua@univen.ac.za,bassey.isong@univen.ac.za)

Abstract - *The quality of software depends on its maintainability. Software under evolution is modified and enhanced to accommodate new requirements. As a result, the software becomes more complex and deviates from its original design, in turn lowering the quality. Program Slicing and refactoring as program re-engineering approaches, makes software systems maintainable. Effective slicing and refactoring requires proper metrics to quantitatively establish the improvement in the quality. Software metrics have proven to reflect software quality, and thus have been widely used in software quality evaluation methods. The results of these evaluation methods can be used to indicate which parts of a software system need to be reengineered. The reengineering of these parts is usually performed using refactoring and slicing - meaning refactoring and slicing are consciously used as a means to improve the quality of software. Probably the most influential factor for software quality is its design. A good design will allow a software system to evolve with little effort and less cost. Therefore by evaluating the quality of the design of a system one can estimate its overall quality. In this paper, we present refactoring and slicing as promising techniques that enhances software quality and the consideration of these techniques when software maintenance is being considered.*

Keywords: software evolvability, software quality, re-engineering, program slicing, refactoring, metrics.

1 Introduction

In recent times, software engineering field has witness an explosion of software systems in diverse working environments including manufacturing and construction industries, information and healthcare services and financial institutions. These evolutions experienced in software systems have made it vital that such systems are useful – functional, flexible, available continuously and operate correctly [1]. In holding to these usefulness criteria during the system lifetime, changes are inevitable as this may be necessary to satisfy requests for performance improvement, functional enhancement, or to deal with errors discovered in the system [2, 3]. Software is often modified to reflect new functionality with the changes of its specification. But we understand that changes propagate and therefore, managing and controlling of these changes posed one of the greatest challenges faced by software engineers [1]. We confirm these facts by the time spent and effort required keeping software systems operational after release. That is, when software systems under evolution are improved, modified, and adapted according to new requirements, the underlying code grows more complex and deviates from its original design, hampering additional modifications and in turn lowering the quality of the software as a result of defect in design [1, 2]. This result in major part of the total software development cost being devoted to software maintenance. Software systems based on large chunk of code are subject to careful maintenance, convenient comprehension, and quality of operation, reliable testing and debugging [9]. Design defects are known to cause the system to display high internal complexity, shocking behaviour, and poor maintainability [4, 8]. Lower reliability of software is due to higher complexity of software. It is therefore necessary to detect and correct design defects to make software maintainable. To cope with this inherent complexity of evolving software, developers need tools and techniques to address these issues.

A promising technique geared towards reducing maintenance effort and enhancing efficient and effective software evolution is program slicing. By way of definition, Program slicing is a technique which extracts all statements and predicates that are likely to affect a certain set of variables in a program. That set of all extracted statements is now known as a program slice. For many years now, research into program slicing technique and its application have been carried out obtaining lots of academic results as well as developed many practical applications [5, 6]. Recent application of program slicing is in program analyzing, understanding, optimizing, debugging, testing, measuring, reusing, transforming, model checking, and software security, maintenance, reverse and reengineering [6, 10]. As an example, program slicing facilitate program understanding thereby reducing the efforts required to perform debugging by efficiently localize faults in a program

[9]. It reduces human efforts in program understanding by focusing one's attention on the statements relevant to some concerned computation. Many techniques for program slicing exist aiming at improving the efficiency of program understanding. Such techniques include static slicing, dynamic slicing, quasi slicing, simultaneous slicing, and conditioned slicing [5, 10]. These techniques employ either tool profiled or user provided runtime information to conduct program slicing.

Program refactoring is another unique technique that makes software systems maintainable, particularly object oriented software. It reduces the complexity found in object-oriented software by incrementally improving the internal software quality without affecting the external behaviour [2, 4]. For software under evolution, refactoring is used to improve the software quality, in terms of, maintainability, complexity, reusability, efficiency, and extensibility [2]. Therefore, it's undoubted to note that software refactoring can perk up software quality in terms of different abilities of software. However, successful refactoring depends on proper metrics which are used to detect design defects and to ascertain quantitatively the improvement in quality after refactoring. If design defects are removed, the complexity will decrease and maintainability will improve, which indicates improvement in quality of software. Hence metrics plays a vital role in refactoring and are tools to quantify particular characteristics of software systems [12]. They provide information to support managerial decision-making during software life cycle [4]. Active research is being carried out with respect to metrics for refactoring and there is abundance of available tools for collecting software metrics and performing automated refactoring on source code thereby simplifying maintenance.

In this paper we investigate program slicing and refactoring as best reengineering practices in maintaining software system and improve quality. Also how software metric can be used to measure the evolvability of software systems**.** These are shown in subsequence sections that follow.

2 Background Information

Usually, before implementing any change, it is vital to understand the software product as a whole and the programs affected by the change in particular [1]. However, program understanding consumes a significant proportion of maintenance effort and resources [1, 10]. Typically, a program is known to execute a large set of functions/outputs. To avoid trying to comprehend all possible program's functionalities, programmers tends to pay attention and understand selected functions (outputs) and those parts of a program that are directly related to that particular function with the goal of identifying which parts of the program are relevant for that particular function in a form that promotes understanding [10]. Program slicing a reverse engineering technique can be utilized to do this.

Program slicing offers a considerable support during program understanding. It is a program decomposition approach that transforms a large program into a smaller one that contains only statements relevant to the computation of a selected function [6]. That is, it captures the computation of a chosen set of variables/functions at some point (static slicing) in the original program or at a particular execution position (dynamic slicing). With this computation, a simplified version of the original version of the program is obtained thereby improving the efficiency of the program understanding and reduces the maintenance effort.

With the program slicing techniques that exist, static slicing methods is known to produce too large slices which are far from really improving the efficiency of program understanding. To overcome this problem, many alternative program slicing techniques have been proposed apart from dynamic slicing, such as Quasi-static slicing, Simultaneous Dynamic Slicing, conditioned slicing [10, 14, 15], and so on. These are considered in subsequent sections.

2.1 Static slicing

Static program slice as defined by Weiser **[5],** is considered as any executable subset of program statements which preserves the original program's behaviour at a program statement for a subset of program variables. In accordance with the original definition of [5,] the slice is defined formally based on slicing criterion $\boldsymbol{C = <p, V>}$ where $\boldsymbol{p}$ is a program point and $\boldsymbol{V}$ is a subset of program variables. A program slice on the slicing criterion $\mathbf{C=<}\boldsymbol{p, V}\mathbf{>}$ is a subset of program statements that conserve the behaviour of the original program at the program point $\boldsymbol{p}$ with respect to the program variables in $\boldsymbol{V}$ [5, 7, 10, 11]. That is, the values of the variables in $\boldsymbol{V}$ at program point *p* are the same in both the original program and the slice. The name static slicing is used to differentiate between the behaviour of the original program that has to be preserved on any input and from other forms of slicing that require the behaviour be preserved on a subset of input to the program [10].

Static slices are worked out by finding sets of indirectly important statements, according to data and control dependencies. It may be used to identify parts of the program that potentially add to the computation of the selected function for all possible programs inputs. Some of the major characteristics of static slicing are as follows. Due to the static nature of the source code analysis, this technique is rather low-cost with respect to overhead and utilization of system resources **[7]**. Additionally, it helps to gain a general understanding of the overall program dependencies of the selected function/variable at a point of interest that contribute to the computation to the selected function. However, though static slicing has many advantages in the process of program understanding, there are still large subprograms that require huge human effort to comprehend due to the imprecise

computation of these slices. As such, static slices cannot be used in the process of understanding of program execution. Also, static slicing has limitations with respect to the accurate handling of dynamic language constructs (like arrays, polymorphism, pointers, aliases, etc.) and conditional statements where its algorithms have to make conservative assumptions with respect to these language constructs resulting in larger program slices. In order to overcome this pitfall, alternative program slicing techniques are of the essence.

2.2 Dynamic Slicing

Dynamic program slicing is one of the alternative program slicing techniques basically designed to overcome the shortcomings of static algorithms through the utilization of the actual program flow information for a particular program execution. With this, dynamic and conditional language constructs can be more precisely handled and therefore produce smaller program slices as against static slicing [5, 6]. The fact is that, a static slice very often contains statements which have no relevance on the variables of interest's values for the particular execution in which the inconsistent behaviour of the program was discovered.

In dealing with the situation, Korel and Lasky **[6]** offered an improvement of static slicing, called dynamic slicing**.** The approach uses dynamic analysis to discover all but only the statements that affect the variables of interest on the particular inconsistent execution. In this way the size of the slice can be considerably reduced, giving way for a better understanding of the code and easier localization of the bugs. Korel and Lasky in **[**6**]** formally defined a slice for a slicing criterion of program ***P*** executed on program input ***x*** as a tuple ***C*** = <***x, vq***> where ***vq*** is a variable at execution position ***q***. A dynamic slice of program ***P*** on slicing criterion ***C*** is any syntactically correct and executable program ***P'*** that is obtained from ***P*** by deleting zero or more statements [6, 7]. The program ***P'***, executed on program input ***x*** generates an execution trace ***T'x*** for which there exists the corresponding execution position ***q'*** such that the value of ***vq*** in ***Tx*** equals the value of ***vq''*** in ***T'x***. That is, the dynamic slice ***P'*** preserves the value of ***v*** for a given program input ***x***.

Several algorithms for dynamic slicing exist and most of them use data and control dependencies to compute dynamic program slices. One of the major needs of dynamic slicing is that it is necessary to identify relevant input conditions for a given program execution for which a dynamic slice should be computed. A generally used approach to identify such input conditions is referred to as an operational profile a concept frequently applied in testing and software quality assurance [9]. Dynamic slices are commonly much smaller than static slices and dynamic slicing may be used to understand program execution during debugging and data-flow testing [7].

3 Dealing with Function Behaviours

Considering the mode of operation of the static and dynamic slicing approaches that considered subsets of the program with respect to either all possible executions or just one execution respectively, a program function can behave in different ways depending on particular inputs to the program. For instance, static slicing techniques might isolate too large code components which include all the possible behaviours of the function thereby producing large slice. On the other hand, dynamic slicing can only identify a code portion with respect to one input which produced slice that could be too small to be generalized and considered as the code portion implementing the desired behaviours.

In order to accommodate the different behaviours of functions that were not solved by static slice and dynamic slice approach, alternatives approaches have been proposed. These are discussed as follows.

3.1. Quasi-Static Slicing

The notion of slice that falls between static and dynamic slices was proposed by Venkatesh **[15]**. This is called quasi-static slice. The need for quasi-static slicing originates from applications where the value of some input variables is fixed while the behaviour of the program must be analyzed when other input values vary. Certainly, a quasi-static slice preserves the behaviour of the original program with respect to the variables of the slicing criterion, on a subset of the possible program inputs [10, 15]. This subset is specified by the possible combination of values that the unconstrained input variables might assume. Of course, in the case all variables are unconstrained, the quasi-static slice corresponds to a static slice, while when the values of all input variables are fixed, and the slice is a dynamic slice.

The concept of quasi static slicing is closely related to partial evaluation or mixed computation [10], a technique applied to specialize programs with respect to partial inputs. Through the specification of the values of some of the input variables, constant propagation and simplification can be used to reduce expressions to constants. With this, the values of some program predicates can be evaluated, thus allowing the deletion of branches which are not executed on the particular partial input. The resulting reduced program is easier to understand than the original one, because details which are not interesting for the particular computations under consideration are eliminated. Quasi static slicing has been applied for program comprehension together with transformations [15]. Quasi static slices are computed on specialized programs.

3.2. Simultaneous Dynamic Slicing

Hall [14] introduced a different form of slicing, simultaneous dynamic program slicing that computes slices with respect to a set of program executions. This approach is so called because it extends dynamic slicing and simultaneously applies it to a set of test cases, rather than just one test case which produces executable slices that are correct on only one input [10, 14]. A simultaneous program slice on a set of test cases is not simply given by the union of the dynamic slices on the component test cases since the union does not maintain simultaneous correctness on all the inputs. Furthermore, Hall [14] proposed an iterative algorithm that, starting from an initial set of statements, incrementally builds the simultaneous dynamic slice, by computing each iteration, a larger dynamic slice.

This approach has been used to locate functionality in code. The set of test cases can be seen as a kind of specification of the functionality to be identified. Combining slicing with this approach results in a more precise identification of the functionality to be extracted. However, while these approaches are cost-effective, very practical and easy to implement and use, they are only good to find components that are unique to a particular functionality [10]. Generally, approach lacks in precision, because the software component identified could be too large and include more functionalities than the one sought.

3.3. Conditioned Slicing

In order to identify program slices corresponding to any function behaviour, a more general model which allows any initial state of the program to be specified is needed. This can be achieve using a first order logic formula which maps a subset of the input program variables onto a set of initial states to the program. The slice resulted by adding such a condition on the input variables to the slicing criterion is called conditioned slice. Conditioned slicing is able to identify slices with respect to any subset of program executions.

It is a general framework for statement deletion based slicing [16]. It allows a better breakdown of the program giving human readers the opportunity to analyze code portions with respect to different perspectives. Conditioned slicing criterion can be specified to obtain any form of slice. A conditioned slice can be computed by first simplifying the program with respect to the condition on the input and then computing a slice on the reduced program. The reduce program can be computed using a symbolic executor called conditioned program [10, 16]. Conditioned slices can be interactively computed as well as automated.

Furthermore, different variants of conditioned slicing have been proposed [16]. Among them are condition-based slices [17] and constrained slice [18]. Also an extension of conditioned slicing, backward conditioning, has been proposed [16]. Application areas of conditioned slicing are on program comprehension and the extraction of reusable functions. As an extension of this approach, the use of symbolic execution to specialize generalized software components to more specific and efficient functions to be used under more restricted conditions has also been proposed [16].

4 Program Slicing and Software Evolution

It is generally accepted that software systems continue to evolve during their lifetime and the long-term success of such a system directly depends on its ability to evolve in response to environmental changes [11]. In order to effect these changes, program comprehension is recommended. The comprehension of source code plays a major role during software maintenance and evolution. However, as software development process matures and grows, it becomes vulnerable to comprehension bottlenecks. That is, understanding a large software product is known as a very challenging task. For instance, debugging large and complex software systems requires a lot of effort since it is cumbersome locating and identifying faults. Therefore, reducing the effort required is an important step towards efficient software evolution [7].

One approach to achieving this is by program slicing. This approach reduce the amount of data to be observed and inspected by focusing and comprehending selected functions (outputs) and those parts of a program that are directly related to that particular function rather than all possible program functions [9]. Program slicing can transforms a large program into a smaller one, cut down the human efforts in program understanding by focusing one's attention on the statements relevant to some concerned computation. In this regard, program slicing becomes relevant to the evolution of programs because it provides a means of evaluating the implications of changing any line of code in that program.

As discussed above, some program slicing techniques have been developed specifically to enhance the quality of software system maintenance work. They make it possible for the maintainers to identify, for instance, the ripple effects of a program change and thus cut-down errors being introduced into the program during maintenance. The underlying principle of program slicing reflects many important features of software evolvability. Program slicing has found application in several evolution processes such as debugging, regression testing, reverse engineering, and so on [8,10]. Also available is tool support to identify slices and enable the localisation of code examination to those parts of the code which need modification and to reflect knock-on, ripple effects [9]. Also, it can be used to measure the cohesion of a program segment [3].

5 Software Refactoring

Most object-oriented software systems in the real-world are developed under evolutionary process models. Change to original source code in software systems has become an important aspect in software maintenance, and refactoring stands as one of the most widely adopted kinds of source code change [4]. Therefore, refactoring is considered as the process of internal enhancement of software without change to externally observable behaviour [12]. For the internal structure, performance or scalability of software systems to be enhanced, changes may be required that preserve the observable behaviour of systems. The major feature of refactoring is improving the internal structure of software system, while not altering the external functionality and behaviour. It is the revision of code in order to make it more efficient, maintainable and easier to work with. It is about 'improving the design of existing code' and as such, it has been practised as long as programs have been written. Refactoring has the benefit of not introducing (or remove) bugs or invalidates any tests, while changes in functionality are disentangled from structural reorganisations [2].

There are three phases involved in the process of refactoring as identify by [4, 21]. These include: (i) identification of when an application should be refactored, (ii) proposing which refactoring technique should be applied and, (iii) where the application of the selected refactoring is to take place. Refactoring is a technique that plays an important role in reducing complexity in software. There are many different refactoring methods that exist, each having a particular purpose and effect. Nevertheless, the effect of refactoring methods on software quality attribute may vary [20]. Also, tool support for refactoring process exists such as fully-automated [21] and semi-automated refactoring tools. Refactoring has been identified as vital to agile (XP) software maintenance and development processes within the software engineering and object-oriented programming communities. It is an increasingly practiced method in industrial software development.

6 Refactoring and Software Evolution

Like program slicing, refactoring plays a vital role in evolution of software systems. As software is enhanced, modified, and adapted to new requirements, code becomes more and more complex at the expense of evolution and maintenance [4, 20]. Therefore coping with the inherent complexity involves improving the internal program structure through a technique called refactoring. A primary goal of refactoring is the minimization of code duplication which usually occurs from copy-and-modified operations. Several refactoring methods that exist such as Extract Method, detailed analysis of function call relations and origin analysis **[22]** are used to facilitate maintenance process.

Other methods employed are data mining techniques and heuristics for detecting refactorings by calculating metrics over successive versions of a system [21]. All these methods fall under the name of refactoring detection in detecting refactoring between different revisions of a software system. This technique is useful for examining the evolution of a system though it does not provide the rationale that led the developers to perform such changes.

7 Refactoring Relation to Metrics and Software Quality

One of the most important techniques of transforming a piece of software in order to improve its quality is refactoring. Refactoring is directly related to metrics and quality of software systems. Software metrics have been confirmed to mirror software quality, and hence have been broadly used in software quality evaluation methods [19]. During the process of reducing the complexity of a software system, the greatest challenge faced by software engineers is discovering where to apply refactoring. Software metrics through evaluation methods can be used to identify which parts of a software system need to be reengineered. Tool support is of the essence in assisting the human intuition in decision-making process in an efficient manner. Refactoring is used to perform the reengineering of these parts. As such, refactoring is deliberately used by developers and software engineers as a means to improve the quality of their software [2, 3].

Software engineering requires quantification, measurement, for effective decision making and quality assurance. Analytical description of process, product and people are needed for a sufficient and visible software development so as to avoid frequent development crisis and inherent complexity of software development. Software metrics give dimension to software management capabilities. It is a vital measure to improve the productivity and quality and as well to quantify particular characteristics of software systems. Software metrics is used to select the order in which refactoring should be performed, taking into account various tradeoffs between code complexity and performance [22].

On the other hand, software quality can be described as the conformance to functional requirements and non-functional requirements (NFRs), which are related to characteristics such as reliability, usability, efficiency, maintainability and portability [22]. By definition, refactoring is used to improve the maintainability of a software product. To identify which refactoring have a positive effect on quality it is necessary to analyze the dependencies between NFRs.

8 Conclusion

Today, senior managers at companies throughout the industrialised world recognized that high quality product

translates to cost savings and an improved bottom line. Even the most jaded software developers will agree that high-quality software is an important goal. But how do we define quality? A wag once said, 'every program does something right, it just may not be the thing that we want it to do.' The problem of quality management is not what people don't know about it. The problem is what they think they do know. Software requirements are the foundation from which quality is measured. Lack of conformance to requirements shows lack of quality. This paper has put forward an argument that forestalls the fact that, the use of refactoring and sliding is an effective and efficient technique for re-engineering practice that is geared towards improving software quality and maintenance.

References

[1] Grubb P. and Takang A.A: Software Maintenance: Concepts and Practice, 2nd ed. World Scientific Publishing Co. Pte. Ltd, Singapore, 2003

[2] K. N. Reddy and A. A. Rao: A QuantitativeEvaluation of Software Quality Enhancement by Refactoring Using Dependency Oriented Complexity Metrics. IEEE Computer Society, 2009

[3] L. Tahvildari, and K. Kontogiannis, "A metric-Based Approach to Enhance Design Quality Through Meta-Pattern Transformations", Proc. of the Seventh IEEE European Conf. on Software Maintenance and Reeng. (CSMR'03), 2003, pp.183-192.

[4] Tom Mens, Tom Tourwe, "A Survey of Software Refactoring", IEEE Trans. Software Eng., vol.30, no. 2, Feb. 2004, pp.126-139

[5] M.Weiser, "Program slicing", *IEEE Transactions on Software Engineering* 10(4), 1984, pp. 352-357.

[6] B.Korel, and J.Laski, "Dynamic program slicing", *In Progrmming Letters*, 29(3), Oct. 1988, pp. 155-163.

[7] J. Rilling and B Karanth: A Hybrid Program Slicing Framework, IEEE, 2002

[8] T.Ishio, S. Kusumoto, and K.Inoue Program Slicing Tool for Effective Software Evolution Using Aspect-Oriented Technique, Proceedings of the Sixth International Workshop on Principles of Software Evolution (IWPSE'03), IEEE Computer Society, 2002

[9] S. Mohsin and Z. kaleem: Program Slicing Based Software Metrics towards Code Restructuring. Second International Conference on Computer Research and Development, IEEE Computer Society, 2010

[10] A. D. Lucia and A. R. Fasolino: Understanding Function Behaviors through Program Slicing. IEEE 1996.

[11] T. Hall and P.Wernick: *Program Slicing Metrics and Evolvability*: an Initial Study, IEEE International Workshop on Software Evolvability (Software-Evolvability'05) Computer Society, 2005

[12] R. Moser, W. Pedrycz, A. Sillitti, and G. Succi: A Model to Identify Refactoring Effort during Maintenance by Mining Source Code Repositories, Springer, PROFES 2008, LNCS 5089, pp. 360–370, 2008.

[13] Chatters B.W, Lehman M.M, Ramil J.F, and Wernick P, "Modelling A Software Evolution Process", *Software Process: Improvement and Practice*, 5, 2000, 91–102.

[14] R.J. Hall, "Automatic extraction of executable program subsets by siinultaneous program slicing", *Jour. of Autom. Soft. Eng.*, vol. 2, no. 1, Mar. 1995, pp. 33-53

[15] G.A. Venkatesh, "The semantic approach to program slicing", *ACM SIGPLAN Notices*, vol. 26, no. 6, 1991, pp. 107-119.

[16] A. D. Lucia Program Slicing: Methods and Applications, IEEE, 2002

[17] J.Q. Ning, A. Engberts, and W. Kozaczynski, "Recovering reusable components from legacy systems by program segmentation", *Proceedings of 1st Working Conference on Reverse Engineering*, Baltimore, Maryland, U.S.A., IEEE CS Press, 1993, pp. 64-72.

[18] J. Field, G. Ramalingan, and F. Tip, "Parametric program slicing", *Conference Record of the 22nd ACM Symposioum on Principles of Programming Languages*, San Francisco, California, U.S.A., 1995, pp. 379-392.

[19] J. Schneider, R. Vasa, and L. Hoon: Do Metrics Help to Identify Refactoring? ACM , Antwerp, Belgium IWPSE-EVOL '10, September 20–21, 2010

[20] E. Mealy and P. Strooper: Evaluating software refactoring tool support. Proceedings of the 2006 Australian Software Engineering Conference (ASWEC'06), IEEE Computer Society, 2006

[21] T. Tourw´e and T. Mens. Identifying refactoring opportunities using logic meta programming. In *Proceedings of 7th European Conference on Software Maintenance and Reengineering*, pages 91–100. IEEE Computer Society, 2003.

[22] K. Stroggylos, D. Spinellis: Refactoring – Does it improve software quality? Fifth International Workshop on Software Quality (WoSQ'07). IEEE Computer Society, 2007

[23] Dae-Kyoo Kim: Software Quality Improvement via Pattern-Based Model Refactoring. 11th IEEE High Assurance Systems Engineering Symposium, Computer Society, 2008World Scientific Publishing Co. Pte. Ltd, Singapore, 2003

Runtime Prediction of Software Service Availability

Davide Lorenzoli, George Spanoudakis
Department of Computing, City University London, London, UK

Abstract *This paper presents a prediction model for software service availability measured by the mean-time-to-repair (MTTR) and mean-time-to-failure (MTTF) of a service. The prediction model is based on the experimental identification of probability distribution functions for variables that affect MTTR/MTTF and has been implemented using a framework that we have developed to support monitoring and prediction of quality-of-service properties, called EVEREST+. An initial experimental evaluation of the model is presented in the paper.*

Keywords: Run-time QoS Prediction, Software Services

1 Introduction

The monitoring of quality-of-service (QoS) properties of software services at runtime is supported by several approaches (see [2] for a survey). Most of these approaches, however, can only detect violations of QoS properties after they have occurred without being able to predict them. This is a significant limitation as the capability to predict violations of QoS properties at runtime is important for the dynamic and proactive adaptation of both the services and the systems that use them. The prediction of QoS properties violations is a challenging problem. This is because there is often a lack of models (e.g., service behavioural models, deployment infrastructure models) that would enable a *white-box* approach to prediction based on analytic techniques (e.g. [21]). Hence, QoS property prediction has often to rely on *black-box* techniques analysing QoS service data collected at runtime.

In this paper, we present a runtime prediction model for a key QoS property of software services, namely software service availability. The model is based on the prediction of two measures of service availability: the *mean-time-to-failure* (MTTF) and *mean-time-to-repair* (MTTR) of a software service, defined as the average up and down time in the operational life of a service, respectively [19][20]. Our MTTR/MTTF prediction model is based on the experimental identification of probability distribution functions for the variables that affect these two measures.

The MTTR/MTTF model has been implemented using an integrated framework that we have developed to support monitoring and prediction of QoS properties of software services expressed in service level agreements (SLAs), called EVEREST+. EVEREST+ enables the monitoring of QoS and behavioural service properties expressed in Event Calculus, and the development and automatic maintenance of runtime models for estimating the probability of potential violation/satisfaction of constraints over such properties (i.e., prediction models).

The development of prediction models in EVEREST+ is based on prediction specifications determining: (a) the key variables underpinning the QoS properties that need to be predicted, (b) the patterns of events that need to be monitored in order to collect runtime data about these variables, and (c) the computations that need to be performed to generate predictions for the properties. Based on these specifications, EVEREST+ can infer automatically the probability distribution functions of the variables that underpin the QoS properties of interest at runtime using monitoring data, and uses these functions to calculate the probability of future violations of the QoS properties.

The rest of this paper is structured as follows. In Sect. 2, we describe the prediction model for software service MTTR and MTTF. In Sect. 3, we describe how the MTTR/MTTF models are realised in EVEREST+. In Sect. 4, we present the results of an initial experimental evaluation of the prediction model. Finally, in Sect. 5 we discuss related work and in Sect. 6, we present concluding remarks and directions for future work.

2 Prediction Model

2.1 Overview

A QoS property prediction in our approach is the computation of the probability that the QoS property will violate a constraint at some future time point t_e, given a request for it received at a time point t_c. Figure 1 illustrates this general formulation of the prediction problem. In particular, t_c in the figure is the time point at which a prediction for a property QoS is requested; t_e is the time point in the future that the prediction is required for; p is the prediction window (i.e., $p=t_e-t_c$); N is the number of QoS values observed between t_s and t_c; Y is the number of future QoS property values that are expected between t_c and t_s; QoS_c is the value of the observed QoS property at the time point t_c; and QoS_y is the value of the predicted QoS property at the time point t_e.

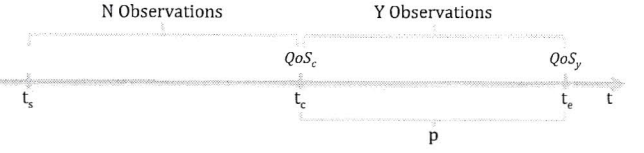


Figure 1. Prediction framework common definitions

Given this generic formulation, the computation of the probability of violating a QoS property constraint is based on estimating the probability of occurrence of different values of variables that affect the QoS property in the prediction window, and can make it violate the constraint. The

probabilities of the values of the underpinning variables are determined by finding the probability distribution functions (PDFs) that have the best fit to sets of historic values of these variables.

2.2 Prediction Model for Software Services MTTR

The overall approach outlined above has been used to develop the prediction model for the *mean-time-to-repair* (MTTR) and *mean-time-to-failure* (MTTF) of a service. More specifically, the MTTR of a software service is defined as the average time from a failure of a service to respond to an operation call until it restarts responding to operation calls normally. MTTR needs to be bounded to ensure the timely reactivation of a service after periods of unavailability. In an SLA, this would be typically specified as a boundary constraint of the form: MTTR ≤ K where K is a constant time measure.

The estimation of the probability of violating the boundary constraint MTTR ≤ K at a future time point t_e is based on identifying the probability distribution functions of two variables: (1) the MTTR of the service of concern, and (2) the time between non-served calls of service operations (i.e., service failures) that occur in a period during which a service has been available (referred to as *time-to-failure* or "TTF" henceforth). MTTR and TTF values correspond to the periods in the operational life of a service shown in Figure 2. More specifically, MTTR is computed as the average of TTR values, i.e., the time difference between the timestamp of the first served call of a service following a period of unavailability and the timestamp of the initial non served call (NS Call) of the service that initiated this period. TTF is the difference between the timestamps of two NS calls of the service that initiate two distinct and successive periods of unavailability.

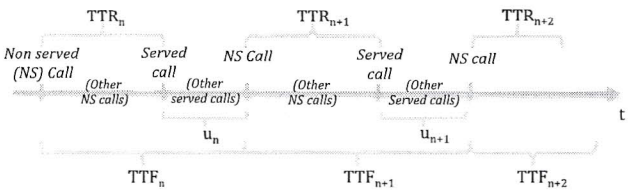


Figure 2. TTR and TTF values

Assuming that N is the number of MTTR values recorded until the time t_c at which the prediction is requested, t_e is the future time point for which the prediction is requested, and y is the – yet unknown – number of TTF values that will be recorded during the prediction horizon p (or, equivalently, the number of cases where the service fails again following a period over which it has been available), to violate the MTTR constraint at t_e the following condition must be false:

$$\mathrm{MTTR_e} = (\mathrm{N}\times\mathrm{MTTR_c} + \mathrm{y}\times\mathrm{MTTR_y})/(\mathrm{N}+\mathrm{y}) \le \mathrm{K} \qquad (1)$$

According to (1), the MTTR value at t_e (i.e., $MTTR_e$) is computed by replacing each of the N TTR–values that have been recorded until t_c by the average value $MTTR_c$ that has been recorded until t_c, and each of the y TTR–values from t_c to t_e by their average value $MTTR_y$ since:

$$\sum_{i=1}^{N} \mathrm{TTR_i} = \mathrm{N}\times\mathrm{MTTR_c} \qquad \sum_{j=N+1}^{N+y} \mathrm{TTR_j} = \mathrm{y}\times\mathrm{MTTR_y}$$

From formula (1), however, it can be deduced that for the MTTR constraint to be violated it must be that:

$$\mathrm{MTTR_y} \ge (\mathrm{K}\times(\mathrm{N}+\mathrm{y}) - \mathrm{N}\times\mathrm{MTTR_c})/\mathrm{y} \qquad (2)$$

Given (2), however, there are two factors to take into account to predict MTTR:

(a) The probability to observe y reactivations of the service in the time period p from t_c to t_e, denoted as $Pr(y)$ in the following, and

(b) The probability of having $MTTR_y > MTTR_{crit}$ (i.e., $Pr(MTTR_y > MTTR_{crit})$) where $\mathrm{MTTR_{crit}} = (\mathrm{K}\times(\mathrm{N}+\mathrm{y}) - \mathrm{N}\times\mathrm{MTTR_c})/\mathrm{y}$.

Based on the probabilities (a) and (b), the probability to violate the constraint MTTR ≤ K at the end of the time period p can be estimated approximately by the following formula:

$$\Pr\left(\bigwedge_{y=1}^{M} E_y\right) = \begin{cases} 1 - \sum_{y=1}^{M} \Pr(y)\times\Pr(MTTR_y \le MTTR_{crit}), & MTTR_c > K \\ \sum_{y=1}^{M} \Pr(y)\times\Pr(MTTR_y > MTTR_{crit}), & MTTR_c \le K \end{cases} \qquad (3)$$

Formula (3) distinguishes two cases: (a) the case where the last recorded MTTR value at the time when the prediction is requested violates the constraint (i.e., the case where $MTTR_c > K$), and (b) the case where the last recorded MTTR value at the time when the prediction is requested does not violate the constraint (i.e., the case when $MTTR_c \le K$). In the former case, the probability of violation is computed as the probability of not seeing a value of MTTR in period p (i.e., $MTTR_y$) that is sufficiently small to restore the current violation. In the second case, the probability of the violation is computed as the probability of seeing a large enough $MTTR_y$ value (i.e., a value greater than $MTTR_{crit}$) that would violate the constraint. However, since the value of y is not known at t_c when the prediction is requested, formula (3) considers values of y up to an upper limit M. The value of M is determined by a condition over its probability. More specifically, M is set to the largest value that makes $Pr(y=M)$ arbitrarily small (i.e., $Pr(y=M) < e$ for an arbitrary small e[1]).

To compute the probabilities $Pr(y)$, $Pr(MTTR_{crit} \le MTTR_y)$, and $Pr(MTTR_{crit} > MTTR_y)$ we need to know the cumulative distribution functions (CDF) for the variables y and $MTTR$, referred to as CDF_y and CDF_{MTTR}, respectively. Given CDF_{MTTR}, $Pr(MTTR_{crit} > MTTR_y)$ and $Pr(MTTR_{crit} \le MTTR_y)$ can be computed by formulas (4) and (5) below:

$$\Pr(\mathrm{MTTR_{crit}} > \mathrm{MTTR_y}) = \mathrm{CDF_{MTTR}}(\mathrm{MTTR_{crit}}) \qquad (4)$$

$$\Pr(\mathrm{MTTR_{crit}} \le \mathrm{MTTR_y}) = 1 - \mathrm{CDF_{MTTR}}(\mathrm{MTTR_{crit}}) \qquad (5)$$

To compute $Pr(y)$ we do not use CDF_y but the probability distribution function of *TTF* variable (i.e., the difference between the timestamps of two NS calls of the service that initiate two distinct and successive periods of unavailability,

[1] The value of e is set by the requester of the prediction.

as defined earlier). This is because over a time period of p time units TTF is related to y as shown by the formula below:

$$\mathrm{TTF} = \mathrm{p/y} \tag{6}$$

As (6) indicates, TTF is an invertible monotonic function g of y (i.e., $TTF = g(y) = p/y$ where p is constant). Thus, the cumulative probability distribution of y $CDF_y = g(CDF_{TTF})$ can be computed by the formula:

$$CDF_y(y) = 1 - CDF_{TTF}(g^{-1}(y)) = 1 - CDF_{TTF}(p/y) \tag{7}$$

where $g^{-1}(y) = TTF = p/y$.

Furthermore, to compute $Pr(y)$, instead of considering a single value v of y, we consider the range $\mathfrak{N}(v) = (v - 0.5, v + 0.5]$. Thus, assuming that $x_1 = y - 0.5$ and $x_2 = y + 0.5$, $Pr(y)$ is computed by the following formula:

$$Pr(y) = CDF_y(x_2) - CDF_y(x_1) = CDF_{TTF}(p/x_1) - CDF_{TTF}(p/x_2) \tag{8}$$

In the case of MTTF, the typical SLA constraint that should be monitored and forecasted is MTTF ≥ K (the largest the MTTF the less frequent the failures of the given services) and the probability of violating this constraint can be estimated approximately by the formula:

$$\Pr\left(\bigwedge_{y=1}^{M} E'_y\right) = \begin{cases} 1 - \sum_{y=1}^{M} \Pr(y) \times \Pr(MTTF_y \geq MTTF_{crit}), & MTTF_c < K \\ \sum_{y=1}^{M} \Pr(y) \times \Pr(MTTF_y < MTTF_{crit}), & MTTF_c \geq K \end{cases} \tag{9}$$

The above formula is derived similarly to the case of MTTR but due to space restrictions the details of its derivation are omitted (see [24] for a full account).

3 Runtime Computation Of MTTR/MTTF Models

The prediction models of MTTR and MTTF have been implemented using EVEREST+. EVEREST+ is a general-purpose framework for monitoring functional and QoS properties of distributed systems at runtime, and developing prediction models for QoS properties. The framework includes two subsystems: (1) a core monitoring subsystem, called EVEREST (*EVEnt REaSoning Toolkit* [16]), and (2) a prediction subsystem. The monitoring subsystem of EVEREST+ checks functional and QoS properties based on events intercepted from services using internal or external event captors. Whilst monitoring QoS properties, EVEREST stores QoS related information, including computed QoS property values, instances of violations and satisfaction of guaranteed constraints of QoS properties that have been set in SLAs, and the values of any other state variables that might have been taken into account in checking particular QoS properties. These types of information are made available through an API.

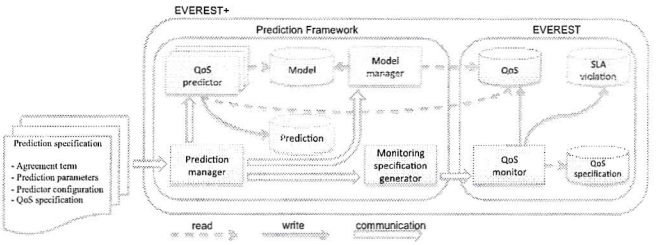


Figure 3. EVEREST+ Architecture

The prediction subsystem of EVEREST+ (see *prediction framework* in Figure 3) supports the generation of predictions for potential violations of guaranteed constraints for QoS properties upon request. This support is available through generic functionalities including: the automatic fitting of different built-in probability distribution functions *(PDFs)* to different types of historical QoS data generated by EVEREST; the selection of the PDFs that have the best fit with the data, the update of PDFs following the accumulation of further QoS property monitoring data; and the generation of predictions for QoS property violations based on built-in and/or user defined functions making use of the probabilities returned by the fitted PDFs.

EVEREST+ automates the prediction generation process based on *prediction specifications* expressed in an extension of the SLA specification language *SLA** [22] that we have developed for this purpose.

3.1 Prediction Specifications

A prediction specification (PS) includes: (a) a set of *generic parameters* for the forecast, (b) the *agreement term* to be forecasted, (c) a *predictor configuration*, and (d) a *QoS specification*. In the following, we examine these parts of prediction specifications more closely and illustrate them through the example PS shown in Figure 4, which is used to derive the prediction model for MTTR (the grey text within parentheses in the figure shows the differences of the PS for MTTF prediction model).

(i) Generic parameters: The generic parameters in a PS determine: the identifier of the *service* and the *operation* that the QoS property to be forecasted relates to, the *prediction window* of forecasts (i.e., the time period in the future that the prediction is required for), and the *history size* of forecasts (i.e., the size of the historic event set that will be analysed to derive the prediction model). In the example of Figure 4, the PS refers to the operation *Ping* of service *Srv* and sets the prediction window to 10 minutes, and the history size to 500 events.

(ii) Agreement term: The agreement term in a PS specifies the constraint that should be satisfied for the QoS property of interest by the target service and operation of the PS (aka *guaranteed_state*). In the example in Figure 4, the guarantee state refers to the QoS property MTTR (MTTF) and the specified constraint regarding is that MTTR (MTTF) must be less than (greater than) 10 seconds.

(iii) Predictor configuration: The predictor configuration in a PS indicates which prediction model to use for computing the probability of the guarantee state of the PS, and the variables whose probability distribution functions will need to be determined from historical monitoring data as they will be needed by the predictor. The former is specified as the value of the attribute *predictor.id* of the predictor configuration in a PS and the latter are specified by the element *prediction variables* that includes a list of variables, each of which is specified by a name/value pair.

```
1   prediction_specification {
2     generic_parameters {
3       service.id = Srv
4       operation.id = Ping
5       prediction.window.value = 10
6       prediction.window.unit = minute
7       history.window.size = 500
8       history.window.unit = event
9     }
10    agreement_term {
11      guaranteed_state {
12        expression.qos = MTTR (MTTF)
13        expression.operator = less_than (greater_than)
14        expression.value = 10
15        expression.unit = second
16      }
17    }
18    predictor_configuration {
19      predictor.id = MT_SV_PRED
20      prediction_variables {
21        variable {
22          name = EVEREST+.model.distribution
23          value = MTTR (MTTF)
24        }
25        variable {
26          name = EVEREST+.model.distribution
27          value = TTF
28        }
29      }
30    }
31    qos_specification {
32      specification.name = MTTR (MTTF)
33      specification.value = MTTR-Formulas (MTTF-Formulas)
34    }
35  }
```

Figure 4. Prediction specification for MTTR/MTTF

In Figure 4, the predictor configurator identifies the predictor as "MT_SV_PRED" (i.e., a parametric predictor for formulas (3) and (9) – see Sect. III.C). It also identifies the two variables used as parameters for the two models (i.e., "MTTR" and "TTF" for the MTTR model and "MTTF" and "TTF" for the MTTF model). The names used for the identification of the two prediction parameters of the MTTR model correspond to names of fluent-variables used in the operational monitoring specifications of the relevant QoS properties. Based on this convention, EVEREST+ can identify the monitoring data that need to be considered in determining the probability distributions functions of the relevant variables at runtime, as we explain below.

(iv) QoS Specification

The QoS specification within a PS provides the operational monitoring specification of the guaranteed state of the QoS property that the prediction is required for. This monitoring specification is expressed in the Event Calculus based monitoring language of EVEREST+, called *EC-Assertion*. A full description of EC-Assertion is beyond the scope of this paper and can be found in [16]. In the following, however, we provide an overview of the language to enable the reader understand the monitoring specification of MTTR listed below. In EC-Assertion, a guaranteed state over a QoS property is expressed by a *monitoring rule* and a set of zero or more *assumptions*. Both monitoring rules and assumptions in have the general form: `body⇒head`. The semantics of a monitoring rule of this form is that when the `body` of the rule evaluates to `True`, its `head` must also evaluate to `True`. The semantics of an assumption is that when the `body` of the assumption evaluates to `True`, its `head` can be deduced as a consequence. The body and head of EC-Assertion rules and assumptions are defined in terms of the following Event Calculus predicates:

(a) The predicate *Happens(e,t,R(lb,ub))* which denotes that an instantaneous event *e* occurs at some time t with in the time range *R(lb,ub)*;

(b) The predicate *HoldsAt(f,t)* which denotes that a state (a.k.a. *fluent*) *f* holds at time *t*;

(c) The predicates *Initiates(e,f,t)* and *Terminates(e,f,t)* which denote the initiation and termination of a fluent *f* by an event *e* at time *t* respectively; and

(d) The predicate *Initially(f)* which denotes that a fluent holds at the start of the operation of a system.

The QoS specification of the MTTR of a service *_Srv* in EC-Assertion is shown in Table 1. The formulas in Table 1 check whether the MTTR is always below a given threshold *K*. More specifically, the rule R1 checks for the MTTR condition violations when a call of operation _O in service *_Srv* is served after a period of service unavailability. The first two conditions in the rule (see *Happens* predicates) check whether the operation call has been served, i.e., whether the service has produced a response to the call within *d* time units. The third condition of R1 (cf. predicate *HoldsAt(Unavailable(_PeriodNumber,_Srv,_STime),t_1)*) checks whether the served operation call happened at a time when the service has been unavailable, and the fourth condition establishes the MTTR value at the time of the call.

The assumption R1.A1 in Table 1 initiates the fluent *Unavailable(_PeriodNumber+1,_Srv,t_1)* to represent a period of service unavailability. This fluent is initiated when a service call occurs (i.e., the call represented by the event *_id1*) without a response to it within *d* time units, and at the time of the occurrence of the call the service is not already unavailable (i.e., no fluent of the form *Unavailable(_PeriodNumber, _Srv, _STime)* already holds). The assumption R1.A2 terminates the fluent that represents a currently active period of service unavailability (i.e., the fluent *Unavailable(_PeriodNumber, _Srv, _STime)*),when a served service call occurs whilst the service is unavailable. The assumption R1.A3 updates the fluent *MTTR(_Srv, _PeriodNumber, _MTTR)* that represents the mean length of consecutive periods of service unavailability, i.e., the value of the variable *_MTTR* of the fluent (note: The formula for the *MTTRnew* variable in Table 1 is provided separately, to make the specification easier to read. In EVEREST+, this formula should be specified inside the fluent *MTTR(...)* of the last *Initiates* predicate of R1.A3).

Table 1. QoS Specification of MTTR

(1) Rule R1:

$$\begin{aligned}
&\textbf{Happens}(e(_id1, _Snd, _Srv, Call(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\textbf{Happens}(e(_id2, _Srv, _Snd, Response(_O), _Srv), t_2, [t_1, t_1 + d]) \wedge \\
&\exists _PeriodNumber, _STime, MTTR : \textbf{HoldsAt}(Unavailable(_PeriodNumber, _Srv, _STime), t_1) \wedge \\
&\textbf{HoldsAt}(MTTR(_Srv, _PeriodNumber, _MTTR), t_1) \Rightarrow \\
&_MTTR < K
\end{aligned}$$

(2) Assumption R1.A1:

$$\begin{aligned}
&\textbf{Happens}(e(_id1, _Snd, _Srv, Call(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\neg\textbf{Happens}(e(_id2, _Srv, _Snd, Response(_O), _Srv), t_2, [t_1, t_1 + d]) \wedge \\
&\neg\exists _PeriodNumber, _STime, MTTR : \textbf{HoldsAt}(Unavailable(_PeriodNumber, _Srv, _STime), t_1) \wedge \\
&\exists _PeriodNumber, _MTTR : \textbf{HoldsAt}(MTTR(_Srv, _PeriodNumber, MTTR), t1) \Rightarrow \\
&\textbf{Initiates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), Unavailable(_PeriodNumber + 1, _Srv, _STime), t_1) \wedge \\
&\textbf{Terminates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), MTTR(_Srv, _PeriodNumber, _MTTR), t_1) \wedge \\
&\textbf{Initiates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), MTTR(_Srv, _PeriodNumber + 1, _MTTR), t_1)
\end{aligned}$$

(3) Assumption R1.A2:

$$\begin{aligned}
&\textbf{Happens}(e(_id1, _Snd, _Srv, Call(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\textbf{Happens}(e(_id2, _Srv, _Snd, Response(_O), _Srv), t_2, [t_1, t_1 + d]) \wedge \\
&\exists _PeriodNumber, _STime : \textbf{HoldsAt}(Unavailable(_PeriodNumber, _Srv, _STime), t_1) \Rightarrow \\
&\textbf{Terminates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), Unavailable(_PeriodNumber, _Srv, _STime), t_1)
\end{aligned}$$

(4) Assumption R1.A3:

$$\begin{aligned}
&\textbf{Happens}(e(_id1, _Snd, _Srv, Call(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\textbf{Happens}(e(_id2, _Srv, _Snd, Response(_O), _Srv), t_2, [t_1, t_1 + d]) \wedge \\
&\exists _PeriodNumber, _STime : \textbf{HoldsAt}(Unavailable(_PeriodNumber, _Srv, _STime), t_1) \wedge \\
&\exists _PeriodNumber, _MTTR : \textbf{HoldsAt}(MTTR(_Srv, _PeriodNumber, _MTTR), t_2) \Rightarrow \\
&\textbf{Terminates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), MTTR(_Srv, _PeriodNumber, _MTTR), t_2) \wedge \\
&\textbf{Initiates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), MTTR(_Srv, _PeriodNumber, MTTR_{new}), t_2) \\
&\text{where } MTTR_{new} = \frac{_MTTR(_PeriodNumber - 1) + (t_1 - _STime)}{_PeriodNumber}
\end{aligned}$$

A QoS specification must also include a specification of the monitoring pattern that is used to record the values of the prediction variables, which are used in the prediction model of the relevant PS but are not required for the pure monitoring of the property. An example of such a variable is TTF since it is required for predicting the probability of future MTTR values but it is not necessary for monitoring MTTR itself. Hence, the QoS specification of MTTR includes a specification enabling the monitoring of TTF values. Table 2 shows the assumption used to initiate fluents for keeping TTF values (R1.A4).

Table 2. QoS Specification of MTTR formulas for TTF

(5) Assumption R1.A4:

$$\begin{aligned}
&\textbf{Happens}(e(_id1, _Snd, _Srv, Call(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\neg\textbf{Happens}(e(_id1, _Snd, _Srv, Response(_O), _Srv), t_1, [t_1, t_1]) \wedge \\
&\neg\exists _PeriodNumber1, _STime : \textbf{HoldsAt}(Unavailable(_PeriodNumber1, _Srv, _STime), t_1) \wedge \\
&\exists _PeriodNumber2, _TTF, _ITF : \textbf{HoldsAt}(TTF(_PeriodNumber2, _Srv, _TTF, _ITF), t_1) \Rightarrow \\
&\textbf{Terminates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), TTF(_PeriodNumber2, _Srv, _TTF, _ITF), t_1) \wedge \\
&\textbf{Initiates}(e(_id1, _Snd, _Srv, Call(_O), _Srv), TTF(_PeriodNumber2 + 1, _Srv, t_1 - _ITF, t_1), t_1)
\end{aligned}$$

3.2 Computation of MTTR/MTTF prediction models

The MTTR/MTTF prediction models are computed by EVEREST+ dynamically at runtime based on the predictor identified in their PSs. This predictor is identified as *MT_SV* in Figure 4. *MT_SV* is a parametric predictor realising the following formula:

$$\Pr\left(\bigwedge_{y=1}^{M} E^{'}{}_{y}\right) = \begin{cases} 1 - \sum_{y=1}^{M} \Pr(y) \times \Pr(MT*_y \odot MT*_{crit}), & MT*_c \neg\odot K \\ \sum_{y=1}^{M} \Pr(y) \times \Pr(MT*_y \neg\odot MT*_{crit}), & MT*_c \odot K \end{cases} \quad (10)$$

(10) is a parametric form of formulas (3) and (9) for estimating the probability of a QoS constraint of the form $MT* \odot K$ where $MT*$ is the mean time variable of interest, y is a parameter affecting it, $\odot$ is the relational operation used in the constraint (e.g., $<$, $>$, $\leq$, $\geq$), $\neg\odot$ is the negation of this operation, $MT*_c$ is the value of the mean time variable at the time of the prediction request, and $MT*_{crit}$ is the critical boundary value that is determined by the different values of y ($MT*_{crit} = (K \times (N + y) - N \times MT*_c)/y$ as discussed earlier).

When a prediction request is received, EVEREST+ determines the PDF of each of the prediction variables in the PS specifications of MTTR and MTTF (i.e., MTTR, TTF for MTTR and MTTF, TTF for MTTF). This is based on computing the parameters of the alternative PDFs in its built-in PDF function set, and then selecting the one that has the best fit with the last N recorded values of each of these prediction variables (N is the value of the *history.window.size* variable in the prediction specification). The fit of each of the built-in PDFs is measured by the non-parametric *Kolmogorov-Smirnov* (K-S) goodness-of-fit test [18], and the probability distribution that has the smallest goodness-of-fit (GoF) value according to the test is selected for each PS variable.

EVEREST+ has built-in implementations of 43 continuous PDFs. The set of these functions can be extended, provided that new PDF implementations adhere to the fixed interface required by EVEREST+ for such functions.

4 Experimental Results

4.1 Evaluation of precision and recall

To evaluate the precision and recall of the MTTR and MTTF predictor models, we used monitoring data generated from the invocation of the Yahoo *WebSearchService* [23]. Through a Java client that we developed to invoke this service, we collected a total of 5500 invocation and 5500 response events. The service response time in these invocations varied between 800 and 5251 milliseconds (ms) with an average of 1146 ms. In the experiments, we considered all invocations with a response time of more than 1000 ms as "non served" service calls ("failures") (see [17] for a similar definition of failures). Based on this filtering, we obtained 1075 "non served" service operation calls and used them to compute MTTR, MTTF, TTR and TTF values.

The total time range of the 5500 invocations was divided in 9 sub-ranges of equal distance and for each of them we computed the $MTTR_c$ and $MTTF_c$ values for the end of the sub-range. We also used five different QoS constraints for MTTR and MTTF, based on different K values. The K values were determined by the MTTR and MTTF values at the end time point t_c of each of the nine sub-ranges as: $0.75 \times MTT*_c$, $MTT*_c - 1$, $MTT*_c$, $MTT*_c + 1$, $1.25 \times MTT*_c$. For each K, we generated predictions using combinations of different prediction window sizes (i.e., 1, 10, 60 and 600 seconds) and different history sizes (i.e., 100, 300 and 500 data points). Hence, we carried out 540 predictions for each of MTTR and MTTF. The precision and recall of these predictions were measured by the following formulas:

$$\text{Precision} = (TP + FN)/(TP + FP + TN + FN) \quad (10)$$

$$\text{Recall} = TP/(TP + TN) \quad (11)$$

In these formulas, *TP* is the number of true (i.e., correct) positive predictions of QoS constraint violations; *FP* is the number of false (i.e., incorrect) predictions of QoS constraint violations; *TN* is the number of true predictions of QoS constraint satisfaction, and *FN* is the number of false predictions of QoS constraint satisfaction. The criteria for classifying a prediction as a TP, FP, TN or FN are summarized in Table 3.

Table 3. Criteria for TP, FP, TN and FN predictions

	Positive: Probability of QoS constraint violation ≥0.5	**Negative:** Probability of QoS constraint violation < 0.5
True: QoS constraint violated at: $t_c + p$	TP	TN
False: QoS constraint satisfied at: $t_c + p$	FP	FN

We also investigated the effect on precision and recall of: (a) the size of the historic event set (HS) that was used to generate the QoS prediction model, (b) the size of the prediction window (PW), and (c) the K-S goodness of fit measure (GoF) of the probability distribution functions that underpin the prediction model.

Table 4. MTTR and MTTF precision and recall

		MTTR		MTTF	
		Precision	**Recall**	**Precision**	**Recall**
Prediction window (seconds)	**1**	0.96	0.94	0.90	0.93
	10	0.81	0.71	0.79	0.78
	60	0.77	0.61	0.60	0.63
	600	0.47	0.39	0.56	0.67
History size (events)	**100**	0.74	0.67	0.65	0.61
	300	0.75	0.60	0.72	0.83
	500	0.76	0.58	0.77	0.83
Goodness of fit	**0.09**	0.70	0.69	0.75	0.76
	0.18	0.77	0.59	0.69	0.75
	0.27	0.76	0.65	0.75	0.89
	0.36	0.73	0.52	NaN	NaN
Overall		0.75	0.63	0.65	0.70

The recorded recall and precision measures for MTTR and MTTF are shown in Table 4, grouped by the history size, the prediction window and the GoF measure for the best underpinning PDF of MTTR and MTTF. As shown in the table, the overall precision and recall of predictions for all different combinations of HS and PW were 0.75, 0.63 for MTTR and 0.65, 0.7 for MTTF, respectively.

Precision and recall improved significantly for both models when considering short prediction periods, raising to 0.96 and 0.94 for MTTF and 0.9, 0.94 for MTTR when the prediction window was set to 1 sec. This was expected since as the prediction window gets longer, historic data become less indicative of what might happen at the window end.

As shown in Table 4, the precision and recall of the MTTR and MTTF prediction models were not influenced by the history size and the GoF measure. For the history size, this was expected as HS was sufficiently large (1%, 3% and 5% of the total event log). For GoF, the expectation was that a lower GoF would result in higher recall and precision (due to a better fit of the selected PDF to the data) but the experiments did not indicate any effect.

The effect of the prediction window and the history size on the recall and precision of the two models was also tested using a 2-way analysis of variance (ANOVA) for these two factors. ANOVA confirmed that the only statistically significant effect was that of the prediction window on the precision of the MTTR and MTTF predictions. The observed differences in precision for different prediction windows were significant at α=0.01 (the *p*-values for MTTR and MTTF precision where 0 and 0.0019, respectively). ANOVA also confirmed that the history size and the interaction between history size and prediction window had no significant effect in the recall and precision of either model.

4.2 Efficiency

To evaluate the efficiency of the EVEREST+ implementation of the MTTR/MTTF prediction model, we used extended sets of historic events (varying from 50 to 20000 events) and measured the total time that it took to infer the MTTR/MTTF prediction model, i.e., to fit different PDFs to the historic data. Figure 5 shows the time taken by the PDF inference process (i.e., the estimation of the parameters of each PDF from the historic data set) against different sizes of history event sets.

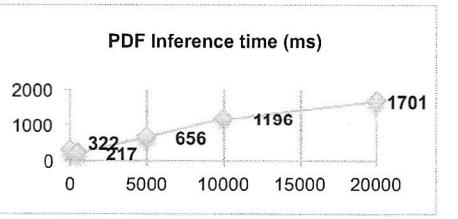


Figure 5. PDF Inference time vs. history event size

As expected, the inference process took longer as the history size became larger. However, increments were linear. Also, since the size of the history event set does not affect the precision and recall of predictions, the MTTR/MTTF models can be efficiently inferred and updated from relatively small historic event sets for which the experiments demonstrated an efficient inference process (e.g., the PDF inference tool 217ms for 500 events).

5 Related Work

Prediction techniques have been developed to generate forecasts for different properties and aspects of software systems. Such techniques may be classified with respect to the property of the software system that a technique aims to predict and the basic algorithmic approach deployed by it (see for a recent survey [13]).

With respect to the prediction property, there have been techniques focusing on prediction of software systems failures [14][8][11], software aging [6], system parameters such as server workloads, CPU loads and network throughput [10][3], or security properties [12]. With respect to the algorithmic approach, different techniques can be characterised as techniques using time series analysis [6][3]; Markov models (e.g. [14][17]); regression models (e.g., weighted regression [4], linear regression [1], auto-regression [5], trend slope estimation [15]); other statistical models (e.g. seasonal Kendall test [7], various mean time prediction techniques [13]); belief based reasoning [12], FSA based prediction [11], machine learning [9] or hybrid techniques (e.g., [8] which uses Markov models and clustering).

Our prediction approach for software services MTTR and MTTF is, to the best of our knowledge, novel both in terms of its algorithmic basis and its focus on prediction of threshold constraints for these availability properties.

6 Conclusions

In this paper, we have presented a prediction model for software service availability based on forecasts of the mean-time-to-repair (MTTR) and mean-time-to-failure (MTTF) for a service. The proposed models realise a black-box where MTTR/MTTF measures are collected at runtime from captured service invocations and responses and are used for the experimental identification of probability distribution functions for variables that affect MTTR/MTTF. The prediction model does not require any behavioural or fault model of the service whose MTTR/MTTF is to be predicted or of the deployment and communication infrastructures used by it. Also the runtime events that are used to infer the prediction models can be collected at the side of either the provider or the consumer of the service.

The MTTR and MTTF prediction models were tested in a series of experiments with positive initial results with regard to the precision and recall of the prediction signals that they generate. The evaluation has also shown the ability to produce and update the prediction models at runtime efficiently.

Currently we are working on developing prediction models for other aggregate QoS properties of software services (e.g., service throughput), without relying on behavioural, compositional or usage service models since such models are scarcely available.

7 Acknowledgments

The work presented in this paper has been supported by the EU Commission; F7 Project SLA@SOI (No. 216556).

8 REFERENCES

[1] Y. Baryshnikov, et al. Predictability of web-server traffic congestion. In *WCW*, 2005.

[2] S. Benbernou et al., State of the art report Gap Analysis of Knowledge on Principles, Techniques and Methodologies for Monitoring and Adaptation of SBAs, Del. #PO-JRA-1.2.1, S-CUBE Project, 2008

[3] N.M. Calcavecchia and E. Di Nitto. Incorporating prediction models in the selflet framework: a plugin approach. In *VALUETOOLS*, 2009.

[4] W. S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 1979.

[5] P. Dinda and D. O'Hallaron. Host load prediction using linear models. *Cluster Computing*, 3(4):265–280, 2000.

[6] S. Garg, et al. A methodology for detection and estimation of software aging. *ISSRE*, pp. 283-292, 1998.

[7] R. O. Gilbert. *Statistical Methods for Environmental Pollution Monitoring*.Van Nostrand Reinhold, New York, NY, 1987.

[8] A. Gunther et al. A best practice guide to resource forecasting for computing systems. *IEEE Trans. on Reliability*, 56(4):615–628, 2007.

[9] Hoffmann, G. A. and Malek, M. Call availability prediction in a telecommunication system: A data driven empirical approach. *25th IEEE Symp. On Reliable Distributed Systems*, 2006.

[10] B.D. Lee and J.M. Schopf. Run-time prediction of parallel applications on shared environments. *IEEE Cluster Computing,* 0:487, 2003.

[11] D. Lorenzoli, L. Mariani, and M. Pezze. Towards self-protecting enterprise applications. In 15th *ISSRE, pp 39--48*, 2007.

[12] D. Lorenzoli and G. Spanoudakis. Detection of security and dependability threats: A belief based reasoning approach. *SECURWARE*, 312–320, 2009.

[13] F. Salfner, M. Lenk and M. Malek. A survey of online failure prediction models. *ACM Comput. Surv.* 42(3), Article 10, 2010.

[14] P. K. Sen. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American Statistical Association*, 63, 1968.

[15] D. Tang and R. K. Iyer. Dependability measurement and modeling of a multicomputer system. *IEEE Trans. Comput.*, 42(1):62–75, 1993.

[16] G. Spanoudakis, C. Kloukinas and K. Mahbub. The SERENITY Runtime Monitoring Framework. In *Security and Dependability for Ambient Intelligence*, Information Security Series, Springer, 2009.

[17] F. Salfner and M. Malek. Using Hidden Semi-Markov Models for Effective Online Failure Prediction. *26th Symp. on Reliable Dist. Systems* 2007

[18] W.T. Eadie, et al. *Statistical Methods in Experimental Physics*. Amsterdam: North-Holland. pp. 269–271, 1971.

[19] D. Chalmers and M. Sloman. A Survey of Quality of Service in Mobile Computing Environments, *IEEE Comm. Surveys*, 2–11, 1999.

[20] E. M. Maximilien, and M. P. Singh,. A Framework and Ontology for Dynamic Web Services Selection. *IEEE Internet Comp.*, 8(5), 2004.

[21] R.H. Reussner, H.W. Schmidt, and I.H. Poernomo. Reliability prediction for component-based software architectures. *J. Syst. Softw.* 66(3), 2003

[22] K. Kearney, F.Torelli, C. Kotsokalis. SLA*:An abstract syntax for service level agreements (2010). Deliverable, F7 EU project SLA@SOI. To be published.

[23] Yahoo WebSearchService. Last visited on 31/12/2011 URL: http://developer.yahoo.com/search/web/V1/webSearch.html.

[24] D. Lorenzoli and G. Spanoudakis. Runtime prediction of MTTR and MTTF violations. Technical report. City University London, 2011

Applying the Theory of Structuration in Enterprise Architecture Design

Dominic M. Mezzanotte, Sr., Josh Dehlinger and Suranjan Chakraborty
Department of Computer and Information Science, Towson University, Towson. Maryland, U.S.A.
{dmezzanotte, jdehlinger, schakraborty}@towson.edu

Abstract *– Enterprise Architecture (EA) defines the infrastructure that acts as a force ensuring alignment of an organization's strategic business plan(s) with its information technology (IT) infrastructure. Current EA frameworks are techno-centric approaches in that business goals, strategies and governance are considered only from the informational aspects of automation. However, EA frameworks do not address the behavioral patterns exhibited by users and the effects of organizational change introduced by EA. Failure to recognize and address negative behavioral patterns often results in the partial or complete abandonment of the EA. This paper extends our approach incorporating ideas from the Theory of Structuration to address human behavior as a significant input to formulating and implementing an EA. The contribution of this paper progresses detailing of human and organizational behavior and utilizing aspects of organizational and behavioral theory to further a framework that focuses on communication as a part of EA development.*

1. Introduction

Enterprise Architecture (EA) is a continually emerging and evolving approach for capturing knowledge about, and documenting, an organization's plan to implement new or modified information technology (IT) [19]. The methodologies used in EA design can range from a very broadly-focused enterprise-wide strategic Information Systems (IS) plan encompassing all aspects of the organization's information and technological needs (the IT infrastructure, hardware, software, and procedures) to one limited in scope and directed at a specific domain and IS application. The product of this effort, an Enterprise Architecture Plan (EAP), expresses an organization's intent and desire to align its strategic business plan with its information technology [14].

The current Enterprise Architecture Framework (EAF) processes rely solely on utilizing the epistemological characteristics, culture and behavioral patterns and policies of the enterprise [9][16]. However, the interactions, goals, objectives, and behavior of stakeholders involved in the development of an EAP are often at odds with the strategic plans of the enterprise resulting in the EAP being only partially implemented or, in many cases, being completely abandoned with many causal factors cited for failure [6][13][17]. Statistics related to EA failure range upwards from 66% to 82% [17] in the private sector to over 84% [4] in public organizations [4][6][17].

Explaining each of the factors associated with failure is often complex and difficult. The complexity arises from the interactions of technology and those tools (software, procedures and processes) required and used to transform collected data into information [4][9][16]. In addition the difficulty in teasing out the individual factors lies in the complex dynamics underlying how stakeholders interact with and react to the technology plan intrinsic to the EA design and implementation.

Formulating and implementing EA represents a significant change in the organizational structure and frequently meets with resistance from stakeholders reluctant to change. Resistance to change is not atypical human behavior. This behavior is often characterized by resultant changes in organizational behavior, usually attributed to management [1][7][11][12], while individuals and organizations historically trend towards a stable environment and a state of equilibrium [9]. In some cases, individual, and consequently organizational, resistance to change may result in work intentionally being sabotaged either overtly or covertly thus allowing the organization to revert to the known equilibrium of the past [11][12].

Where technology is involved, user behavior often reveals itself in the way in which the technology is presented. For example,if the technology is introduced unexpewctedly and without any input from or concern about stakeholders, it then may be either accepted or rejected by those involved in the technological transition [4][9][11][12][16]. When change is unexpected, user resistance to the technology can be considerable [9]. We can conclude that the manner in which the EA design effort is supported and deployed by a development methodology (in the case of EA an EAF), as well as the design and implementation approach taken by organizational management significantly affects how stakeholders perceive and behave during the EA design process [1][7]. Consequently, user behavior should be viewed as the result of, and due to cognitive processes learned resulting from experiencial organizational behavior and their ability to adapt to a changing environment which would in turn affect their respective future capacity for handling and accepting change [1].

The behavior displayed is often in defensive routines and mechanisms such as stakeholder attempts to deceive, manipulate, or distort information during the EA design

process [11][12]. Therefore, user behavior becomes a critical and potentially costly component to EA affecting project success and/or failure with a need to mitigate stakeholder and organizational resistance to change and their respectibe attitudes towards new technology[16].

In the EAF processes used in the major EA methodologies (Zachman Architectural Framework [20], ToGAF [15], FEAF [2], and DoDAF) [1]), the view of user and organizational behavior varies from being either cursorily considered or completely avoided in the EAP process [18][19]. Yet, any discourse regarding technology should be inextricably linked to the social context of its formulation and use. In this context, technology, if taken singularly and in isolation, can be easily constrained resulting in limited stakeholder innovation, creativity, participation, interest, and commitment when designing and implementing the new EA procedures, tasks, techniques, tools, user knowledge requirements, roles, and responsibilities without considering their impact on human behavior [7][11][12][14].

This paper considers both user and organizational behavior as multi-faceted, complex systems/entities that together recursively affect the design, implementation and use of enterprise-wide information technology. The emphasis in this paper explores the benefits of using a socio-communicative based framework that supplements and supports the recursive nature of stakeholder behavior for formulating and designing EA in organizations that builds on earlier work [13].

The remainder of this paper is as follows. Section 2 reviews several EAF methodologies to ascertain how they cope, or fail to cope, with the human and organizational behavioral complexities associated with deployment of enterprise-wide technology exhibited during EA design. Section 3 discusses the *Theory of Structuration* [7] and its potential for application to IT [16] and EA design with a lens to mitigating EA failure. Section 4 briefly describes the preliminary processes of a framework that takes into account user and organizational behavior enhancing existing architectural framework methodologies or its use as a framework by itself. Finally, Section 5 provides a discussion, some concluding remarks, and future research towards an enhanced approach to EA.

2 An Analysis of Current Enterprise Architecture Frameworks and Stakeholder Behavior

In the context of this paper, Project Management (PM) focuses on developing "working software" [5] while EA and the Enterprise Information Architect (EIA) focuses on planning and describing the overall strategic technological solutions and infrastructure which map to the current and future operating model of the enterprise [9][14]. The scope of the EIA's work, therefore, must be wider and include skills related to business acumen as well as sociological and psychological ***soft people*** skills to be effective. Confusing the two activities almost always results in misdirection of effort and a reinforcement of misunderstanding on the part of stakeholders leading to EA failure.

Over the past twenty plus years, several EAF models have been developed and deployed. Included in the following list are five popular EAF methodologies:

- Zachman Enterprise Architecture (Z|FA) [20]
- The Open Group Architecture Framework (TOGAF) [15]
- The Federal Enterprise Architecture Framework (FEAF) [2]
- The Department of Defense Architecture Framework (DoDAF) [1]

The EAF approach taken by each of these unique methodologies provides a framework, a set of tools, used to document and develop a broad range of EA technology-oriented alternatives. The respective collective strengths of each framework lies in their paradigm for documenting the specifications and requirements for IT applications while defining an EAP (an EA blueprint) consisting of and describing a common EA vocabulary, the methods, disciplines, processes, and system and subsystem relationships that communicate the architecture of the EA [20][14][15]. The frameworks impose a disciplined, comprehensive techno-centric regimen of methods, principles, and practices to be used by the EIA resulting in an EAP that allows the organization to then focus on how best to use and align technology with the organization's strategic business plan.

The weakness of each EAF is that the concentration of work is technically oriented and aimed primarily at producing a set of deliverables focused on the technical aspects of EA design specifications, potential modeling schemes, and implementation of the design. The EAP documents the organization's structure, business processes, Information Systems (IS), and related IT infrastructure at a technical, domain-level architecture that fails to take into consideration the impact human behavior hason the work effort. EA should be more holistic and much broader in scope and that it include the technical, business, solution architecture as well the impact of technology on both stakeholder and organization behavior.

In the next section, the recursive nature of EA design is listed as a process to be considered and explored in more depth in EA development. We can demonstrate the importance of the recursive nature of EA design if we consider the elicitation of stakeholder input (specifications) as a critical, vital component of EA. The product from the elicitation process must detail the requisite technical functional and non-functional specifications and requirements (simply specifications) for organizational IT systems and subsystems. The specifications produced provide and are the foundation for the quantitative and qualitative facets of the EA including

implementation, performance, reliability, security, maintainability, governance, and compliance with business strategy. Failure to correctly capture, document, and communicate these specifications back to the stakeholders to verify correctness, often leads to and can result in failed EA design and/or implementation, simply stated "poor architecture".

The specification elicitation process as described is iterative, including modeling of the specification, verification of the specification and written acceptance by the stakeholder (redoing the specification if it is invalid) while keeping the effects of new technology in-line with stakeholder capacity for change. In effect, conforming to the recursive behavior described in Giddens' *Theory of Structuration* [7]. This requires the EIA to maintain a perspective remembering that stakeholder capacity for change is affected by several factors: their past and present organizational experiences, organizational behavior, and what they might perceive and expect in the future. Failure to capture contextual variations in stakeholder perceptions within and across departments within an organization, coupled with failure to communicate and define valid, objective, uniform work flows, processes and information needs can adversely affect EA design.

The regimen imposed by existing EAFS often inhibits innovative, creative problem-solving by limiting stakeholder input typically due to existing organizational rules, procedures, and assumptions [14]. Though the EAFs each provide a mechanism of documenting the EA, there are little or no provisions for incorporating patterns of stakeholder and organizational relationships (i.e., communication) which is essential to monitor and manage human behavior. Communication mechanisms allow management to exercise EA governance and to inform stakeholders of subsequent organizational change caused by the EA. Simply stated, people who will be affected by IT change are unlikely to accept and be impressed by the change if the change is not communicated in a timely effective manner. Therefore, the EA should be predicated upon a communications path, both vertical and horizontal, that recognizes human behavior, the impact of technology, and the actions and relationships of those entities on and within the social structure of the organization. The communications path should foster and encourage a participative EA design that encourages "ownership" by EA stakeholders. In effect, a *socio-communicative* path enhanced with a feedback mechanism that provides a means of maintaining a state of homeostasis offering a channel for stakeholders to exchange ideas, know-how and knowledge benefitting the organization.

3. Structuration and Information Technology

Contemporary EA design pervades the IT world driven by the desire to align business strategy with IT resources with the intent of providing a more efficient and effective organizational environment [14]. The current approach to EA design follows traditional software engineering practices that employ a variety of computer science theories and "computer-oriented functions" that depends heavily on the elicitation of IT specifications based on explicit and tacit organizational knowledge gleaned from stakeholders. These specifications form the foundation for EA which are then transformed into both functional and non-functional design requirements [14][18].

The typical process used by EIAs to progress EA design consists of historical technology-oriented tools/frameworks that include various techniques (methods), procedures and processes, and paradigms designed to achieve and formulate a desired technological solution [18]. However, these technology-oriented tools/frameworks are deficient and fail to recognize and consider the impact of new or enhanced technology on human behavior which can and may significantly affect the success or failure of EA design [16]. This is especially true in the elicitation of "tacit" stakeholder knowledge, that undocumented knowledge known only to a specific individual or group of individuals that may or may not intentionally withhold that information from the EA, which, if not taken into design consideration, can lead to EAs that are only partially implemented or, in some instances, completely abandoned [6][17].

Users of technology (i.e., stakeholders) have historically accepted the infusion of technology (e.g., hardware, software, and processes) solutions into their daily lives without question. That is no longer true today. Stakeholders frequently question, either covertly or overtly, new or enhanced technology in relation to how it might affect the environment in which they function [16]. This change in the human behavior therefore poses a topic of high relevance to EA management and in how EIAs do their work given that new or enhanced technology can be accepted, rejected, or modified to fit the roles desired by stakeholders [16]. It also raises questions about how EIAs must evaluate the quality of the elicited specifications and how the unanticipated use of technology might affect EA design throughout the EA design effort[16]. Stakeholder use of technology therefore manifests itself in two ways:

- How the solution may be either abused or used properly
- Designing the solution such that it identifies and handles abuse.

If we consider the elicitation of stakeholder specifications and subsequent EA requirements as knowledge creation, simply organizational knowledge held by stakeholders, as the intersection and interaction of technology and stakeholder behavior, then how stakeholders create, process, and provide that knowledge as input to the EA becomes a recursive process directly related to organizational behavior and environment. This knowledge is then a dynamic recursive re-conceptualization of organizational knowledge creation that

directly affects the quality of EA design specifications while providing a paradigm for effective EA.

Organizational behavior is based on a set of standardized rules, procedures, processes, and systems (collectively referred to as rules) [11][12]. These rules constitute a set of coordinated and controlled activities with institutional work (i.e., output) produced from complex networks of technical relations that span organizational boundaries [16]. The rules and activities then become an integral part of prevailing social behavior within the organization and are subsequently built into the society as reciprocated interpretations [3][7]. However, conformity to these rules and activities is often problematic as it conflicts with organizational efficiency by undermining, constraining, and inhibiting organizational flexibility. This rule-based environment inhibits and limits a dynamic view of the organization and discourages participation and, in genreal, is not conducive for EA stakeholders to be innovative, creative, and therefore more receptive to change [7][16]. Our approach considers the recursive nature of people and organizations postulated by Giddens' *Theory of Structuration* [7] as a theoretical lens to align EA and IT with a socio-communicative framework that takes into account both stakeholder and organizational behavior.

In the Theory of Structuration, the *duality of structure*, *agency*, *transformation*, the relationship between agency (human actors/agents - stakeholders) and structure (systems/organizations/society), are considered in the following contexts:

- How are the actions of individual agents related to the structural features of organizations/society?
- How do individual agents act on a day-to-day basis?
- How are individual actions reproduced?

The theory posits that structure exists only in and through the activities of human agents and gives form to social life but that that form is itself not the shape of it. Giddens' agency does not refer to people's intentions in doing things but focuses on the behavioral patterns exhibited by people's actions. The duality of structure suggests a social structure consisting of rules and resources with rules being applied to govern and regulate social life and resources including both human and non-human elements that can be transformed into power.

The *Theory of Structuration* examines human actors/agents actions and structures (organization/systems) and is concerned with reworking conceptions of human being and human doing, social reproduction and social transformation. It is based on the premise that human agents or actors operate at both a conscious and unconscious level which constitutes day-to-day behavior and that the routine (i.e., whatever is done habitually) is a basic element of day-to-day social activity (i.e., the recursive and reproductive nature of social life) [7]. It articulates a process-oriented theory that treats structure (institutions) as both a product of, and a constraint on, human action, the result of what actors are able to say about the conditions of their actions and that of others. These discursive phenomena can be detrimental in that human agents/actors may limit or completely avoid to disclose or occlude what they know about what they do and what they say about it. The actions may be intentional or unintentional, conscious or unconscious. Giddens states that all human beings are knowledgeable knowing a great deal about the conditions and consequences of what they do [7]. The knowledgeof human actors is bounded by unconscious action on one hand, and unacknowledged conditions and/or unintended consequences of action on the other. All factors which may affect the veracity of those specifications and requirements needed for quality EA design.

Until recently, the *Theory of Structuration*, has been a theory based on the social sciences, human action, and organizational structure paying little attention to IT. However, the application of the *Theory of Structuration* to IT lends itself as a design tool for EIAs to better understand stakeholder behavior and the conceptual impact on organizational behavior assisting in formulating, designing, and implementing EA technology.

One noted advocate for using the *Theory of Structuration* in IT development and deployment, proposed the *Structurational Model of Technology* (SMT), as a means to understand how technology affects organizations and vice versa [16]. This approach centers on two concepts - the *Duality of Technology* and the *Interpretive Flexibility* of technology [16]. The former posits that the socially created view and the objective view of technology is not exclusive but intertwined and are differentiated because of the temporal distance between the creation of technology and usage of the same. *Interpretive Flexibility* on the other hand defines the degree to which users of a technology are engaged in how it is built and used (physically and/or socially).

From an organizational perspective, institutional properties and practices influence human and consequentially organizational behavior in their respective interaction with technology. Conflicts, tension, and resistance to change can occur as a result of knowledgeable action on the part of human actors. Human actors use technology including: professional norms; rules of use – design standards and available resources (time, money and skills), which acts upon the institutional structure of an organization. The consequence of the institutional interaction with technology manifests itself by impacting the institutional properties of an organization through reinforcing (sustaining) or transforming (changing) structures of signification, domination and legitimization that characterize the institutional realm.

In summary, the theoretical premise of the *Theory of Structuration* [7] and *SMT* [16] acknowledges that

organizational structures, technology and human action are not distinct and disjoint activities/entities but intertwined such that each is continually reinforced and transformed by the other (a recursive process). It considers human behavior as a product of the environment in which they perform and that they are skilled, innovative, and creative performers that carry and create scripts and develop roles that fit the environment and organization in which they function.

Taken together, the *Theory of Structuration* and the *Structuration Model of Technology* highlight those factors which may affect the veracity of those specifications and requirements needed for quality EA design. A logical conclusion can therefore be made that an initiative such as the formulation of enterprise architecture remains incomplete if it does not explicitly take into account human action and organizational behavior from which to derive a dynamic theory of social and institutional order. A solution to this problem can be found in the following section where a where a communications path/channel is established and maintained that encourages knowledge sharing among stakeholders to better ensure and manage EA design.

4. A Socio-Communicative Approach to Enterprise Architecture Design

Many of the emergent causal processes that lead to EA failure has been well documented with those factors of significance attributed to EA failure detailed in our earlier paper [13]. The challenge facing an EIA and the organization is how to avoid these factors and prevent EA failure. The answer to this challenge might lie in the ubiquitous nature of communication, its relevance to human behavior, and its implementation as a modifier for human behavior.

If we consider human behavior from a historical perspective, social theorists posit that organizational behavior directly reflects the attitudes and behavior of top management through all layers of the organization [7][11][12]. In concert with this hypothesis, the addition of technology significantly influences human behavior and the attitudes, feedback loops, and selection mechanisms that typically are used in the EA [16][19]. These then form the basis for an analytical approach wherein socio-communicative processes provide a more meaningful and cogent solution to many EA failure factors.

One of the lead factors and often cited as the major contributor and influence for EA failure, lack of or poor leadership, frequently focuses on the role played by the EIA [9]. However, the manner in which the EA is designed and implemented typically poses at least one counter-productive issue that can be easily mitigated and managed by the EIA: poor communication. In most failed EAs, "poor architecture" is most often attributed as the leading factor for the failure. Upon investigation, the design specifications are identified as the major cause for the failure [4][17]. A correlation between the elicitation of stakeholder specifications, understanding of those specifications, and subsequent transformation of those specifications into EA design requirements can easily be drawn upon review and analysis of the root-cause of failure. However, the problem may go far beyond just poor specifications and requirements.

In most organizations, people behave as the group behaves and tend to use technology in ways in which they are both familiar and understand [3][16]. In addition, their actions and motivations typically are influenced and driven by group behavior [8]. Our goal is to establish an environment where human agents/actors (stakeholders) are willing to share explicit, commonly known and perhaps even documented, knowledge and, more importantly as it may be that needed to ensure EA success, that "tacit" knowledge of what and how they really perform their day-to-day activities. In effect, an environment as stated above that promotes, encourages, and fosters user "ownership" of the EA.

The challenge associated with this goal requires that the EA environment transcend technical and business issues and become more focused on the recursive nature and human aspects of the EA. Human behavior then becomes a prime concern of EA design with communication (the exchanging of information, messages, ideas, opinions, and explicit and tacit knowledge) the motivational mechanism used in clarifying otherwise ambiguous goals and objectives and solving complex IS issues and concerns. This aspect of EA then is just as or more important than any technical and/or business acumen. As the mechanism to progress EA design, communication is an effective learnable skill. To illustrate the communications issue, organizations are faced with three behavioral concerns:

- The introduction of new technology into organizations
- Changes in human behavior resulting from new technology
- Resistance to change

Considering each of these factors, people occasionally resist new technology and change for legitimate reasons:

- A fear of job loss and job security
- A perception of loss of status and responsibility within the organization
- A need to learn new procedures and processes
- A feeling the employer no longer cares about the employee

We propose an alternative solution that puts in place a communications mechanism where an environment exists that fosters and encourages:

- Sharing of ideas that facilitate decision-making and problem-solving

- Explaining decisions and providing an opportunity for clarification
- Sharing responsibility for decision-making and implementation
- Providing specific instructions and closely supervising and rewarding performance (governance)

Our approach establishes, as a first step, EA governance at the beginning of EA design and defines how it will be documented, continued, and maintained throughout the EA life-cycle. The second step requires top-level management to publish a description of what EA is and describing its purpose, the reasons for, and the role EA has within the organization detailing an initial list of EA goals and objectives including the magnitude, scope, and boundaries for the EA. The third step defines and establishes the communications process describing the:

- Format for all EA design and implementation correspondence such as review meetings and the collection of EA specifications
- Mechanism for administering and managing the EA governance
- Procedure for providing input to the EA design for suggesting innovative and creative ideas, opinions, and endogenous and exogenous factors that might influence the decision-making process
- Structure of the communications path including who is to receive copies of EA related correspondence such as news bulletins, interview, and EA related information.

The goal of this proposed framework is to create an architecture that provides best chance for success as well as the most adaptable, practical solution for the future and which aligns strategic business with IT plans.

5. Discussion, Concluding Remarks, and Future Directions

Systems of coordinated and controlled activities result from work embedded in complex networks of technology-centric relations and boundary-spanning exchanges. Existing EAFs do not address the behavioral aspects of humans and the organization resulting from technology. The *Theory of Structuration* provides a means of understanding human behavior and its relationship to organizational change [7]. *SMT*, on the other hand, addresses the effects of technology on human behavior [16]. Taken together, they conceptualize the unique opportunities for an EIA to implement an EA.

This paper explores the possibilities extant and the potential contribution marrying the *Theory of Structuration* with *SMT* in an EA can make towards improving the EAF process. It further initiates the clarification of an ontology and paradigm describing an architectural framework that encourages an environment that fosters stakeholder participation and collaboration in the EA design process and a forum for information-sharing. It allows human agents/actors to communicate whenever and however they need to in order to solve problems and exchange knowhow and knowledge.

The possibility and prospect of a successful EA becomes realizable if an enhanced participative working environment is encouraged and made a part of the design and implementation process that welcomes new EA technology and avoids it being perceived as a risk or threat to stakeholder well–being. At the same time, it provides the mechanism for EA governance documenting the EA from inception throughout its life-cycle. Future work expands the framework clarification above in more depth and detail. The goal of this work is to provide a communications process (a framework in its own right) that can be used in conjunction with existing EAF methodologies with the potential it offers for removing many of the behavioral obstacles which inhibit EA deployment.

References:

[1] No author listed, *The DoD Architecture Framework Version 2.0DoD Deputy Chief Innfromation Officer*, Department of Defense, April, 2010.

[2] Chief Information Officers Council, *Federal Enterprise Architecture Framework*, CIO Council, Version 1.1, August 5, 1999.

[3] E. Davidson, *Technological Frames Perspective on Information Technology and Organizational Change*, The Journal of Applied Behavioral Science, 2006 : 42: 23

[4] C. Ferreira and J. Cohen. "Agile Systems Development and Stakeholder Satisfaction: A South African Empirical Study," *Proceedings 2008 Conference of South African Institute Computer Scientists and Information Technologists on IT Research in Developing Countries*, pp. 48-55, 2008.

[5] J. Gido and J. Clements, *Successful Project Management*, South Western CengAGE Learning, 4th Edition, 2009.

[6] R. Gauld. "Public Sector Information System Failures: Lessons from a New Zealand Hospital Organization," *Government Information Quarterly*, 24(1):102-114, 2007.

[7] A. Giddens. *The Constitution of Society: Outline of the Theory of Structuration*, University of California Press, 1984.

[8] P. Hersey abd K. H. Blanchard, *Nabagement of Organizational Behavio: Utilizing Human Resources*, 5th Edition, Prentice-Hallm Inc., Englewood Cliffs, NJ, 1988

[9] R. Lewin and B. Regine. "Enterprise Architecture, People, Process, Business, Technology," Institute for Enterprise Architecture Developments [Online], Available: http://www.enterprise-architecture.info/Images/Extended%20Enterprise/Extended%20Enterprise%20Architecture3.html.

[10] A. Lindstrom, et al, *A Method to Assess the Enterpriese-wide IT Resource for Performance and Investment Justification*, Dept of Industrial Information Systems and control Systems, Royal Institute of Technology, KTH, SB-100, 44 Sto0ckholm. Sweden, 2005

[11] A. Maslow. *Motivation and Personality*, Harper Collins, 1987.

[12] D. McGregor. *The Human Side of Enterprise*, McGraw-Hill, 1960.

[13] D. M. Mezzanotte, Sr., J. Dehlinger, and S. Chakraborty, *On Applying the Theory f Structuration in Enterprise Architecture Design*, IEEE/ACIS, August, 2010.

[14] D. Minoli. *Enterprise Architecture A to Z*, CRC Press, New York, 2008.

[15] The Open Group, *TOGAF Version 9*, The Open Group, 2009.

[16] W. Orlikowski and D. Robey, *Information Technology and the Structureing of Organizations*, The Institute of Management Science, 1991.

[17] S. Roeleven, Sven and J. Broer. "Why Two Thirds of Enterprise Architecture Projects Fail," *ARIS Expert Paper* [Online], Available: http://www.ids-scheer.com/set/6473/EA_-_Roeleven_Broer_-_Enterprise_Architecture_Projects_Fail_-_AEP_en.pdf.

[18] N. Rozanski and E. Woods, *Software Systems Architecture*, Addison-Wesley Professional, 2006.

[19] C. Seaman, Qualitative Methods in Empirical Studies of Software Engineering, IEEECS, Log Number 109541, June, 1998.

[20] J. Zachman, *Concepts of the Frameword for Enterprise Architecture*, Information Engineering Services, Pty, Ltd., No Date.

An Analysis of Business Agility Indicators in SOA Deployments

M. Hirzalla[12], P. Bahrs[2], J. Huang[1], C. Miller[1], and R. High[2]
[1]School of Computing, DePaul University, Chicago, IL, USA
[2]IBM, Austin, TX, USA

Abstract - Business agility is often claimed as one of the primary benefits of Service-Oriented Architectures (SOA). However neither this claim, nor the effect of individual SOA practices on business agility, has been empirically investigated in a systematic way. In this paper we explore the impact of architectural concerns, business process management, infrastructure support for impact analysis, loose coupling between services, and governance practices upon the achieved business agility of a SOA project. We present a study which reports the technical practices of 39 SOA projects from multiple domains in order to identify technical factors which most closely correlate with business agility. Finally we demonstrate how the findings from our study can be used to increase business agility practices in a SOA project.

Keywords: SOA, Business Agility, Metrics, Software Engineering

1 Introduction

Service Oriented Architecture (SOA) solutions [1] [2] are quickly becoming the new 'norm' in enterprise level projects. In effective SOA deployments information technology and business processes are aligned through the strategic decomposition of functionality into services [2] [3], and the overall system is delivered through a composition of such services. As a result, businesses adopting SOA solutions are often well poised to meet new business challenges and to achieve increased *business agility*.

The term *business agility* has been defined in several different ways. For example, Dove [1] describes it as the ability to "manage and apply knowledge effectively, so that an organization has the potential to thrive in a continuously changing and unpredictable business environment." Similarly, Gartner [2] defines it as "the ability of an organization to sense environmental change and respond efficiently and effectively to that change." Finally, Kidd [4] defines it as the ability of enterprise elements to adapt proactively to unexpected and unpredicted changes. Although emphasizing slightly different concepts, these definitions all focus around the ability of an organization to adapt responsively to changing events.

According to Cummins [5], enterprise level agility can be achieved through incorporating both SOA and business process management (BPM) techniques to improve and optimize the design of business processes and deliver an effective governance structure. He further claims that enterprises that achieve business agility often use model-based tools to "manage complexity, support optimization of enterprise operations, and enable rapid reconfiguration of service units to respond to changing business threats and opportunities." In this paper we investigate these claims through studying SOA adoption practices in 39 different SOA deployments.

Although many different factors have been claimed as contributors to enterprise level business agility the work described in this paper focuses primarily on the more technical aspects of SOA implementations including business process management, architectural considerations, infrastructure support for impact analysis, loose coupling between services, and governance practices related to the SOA deployment. Other factors such as organizational structure, governance policies, social interactions, education, and skills are left for future work.

The results reported in this paper are part of a far broader study that we are conducting in order to construct a statistically validated business agility Index (BAI) [6] designed to predict whether a current SOA project is likely to achieve business agility or not. In this paper, we examine several SOA practices and evaluate the extent to which each of them is present or absent in business agile vs. non-business agile SOA projects. This provides interesting insights into the industrial adoption of commonly advocated SOA practices, and also reports correlations between the various practices and achieved business agility.

The data for this study was collected through an IBM Academy of Technology virtual conference. A call for papers was issued in early May 2010 to request participation with case studies and surveys. The first round of data collection lasted for three months from June through September of 2010, and participants were asked to complete an extensive survey which involved a series of questions that were to be answered with respect to a single SOA project. The survey included a total of 215 questions, out of which 80 questions were used for the purposes of this study. All projects in the study were required to have SOA as the primary architecture style, and to include five or more services. Furthermore, the projects were required to have been production bound with real funding, teams, accountability, milestones, return on investment

measure, and schedules, etc, and also to have completed the full development lifecycle. Projects were included in the study whether or not they achieved business agility.

The remainder of the paper is structured as follows. Section 2 describes the SOA projects that were included in our study, as well as the process used to investigate the projects. Section 3 describes the specific technical factors that we studied. Section 4 reports the overall results for various projects and provides a simple evaluative framework based on the identified factors and visualized using a spider graph. Section 5 summarizes our results and section 6 provides an evaluation of a random SOA project. Finally, section 7 discusses conclusions and future work.

2 SOA Practices

TABLE I
BUSINESS AGILITY FACTORS

Dimension	Description
SOA Architecture (SOA)	Measures the degree to which a project adhered to SOA architectural principles.
Impact Analysis (IA)	Measures the degree to which a project handled change impact analysis.
Business Process Management (BPM)	Measures the degree to which a project adhered to BPM best practices and principles.
Loose Coupling (LC)	Measures the degree to which the solution used practices which contributed towards loose coupling between components.
Governance (Gov)	Measures the degree to which a project handled Governance and its related attributes.

In preparation for the online conference, twenty SOA experts met over a period of eight weeks to define what it means to be business agile. These experts included a mix of IBM fellows, distinguished engineers, senior IT architects, project managers and research staff. All participants were experienced in SOA and its best practices, and the average IT and SOA experience of the experts was approximately 20 and 5 years respectively. The SOA experts were divided into two separate groups, each group was tasked with independently identifying what it means for a SOA project to contribute to business agility. Findings from the two groups were discussed until consensus was reached. The results were then documented in the form of 80 factors that experts agreed were fundamental for achieving business agility. We hypothesized that projects achieving higher degrees of business agility would be more likely to exhibit high scores for the identified factors, and conversely that projects which failed to achieve business agility would be likely to receive lower scores.

TABLE II
BUSINESS AGILITY FACTORS AND THEIR ATTRIBUTES

Composite	Factored Significant Attributes
SOA Score (SOA)	1.1 Architecture support for rules
	1.2 Architecture support for events
	1.3 Architecture support for task automation
	1.4 Architecture support for alerts
	1.5 Architecture support for resource allocation.
	1.6 Architecture support for monitoring
	1.7 Architecture support for analytics
	1.8 Creation of service architecture
	1.9 Maintaining architecture decisions
Impact Analysis Score (IA)	2.1 Architecture supports reporting analysis
	2.2 Requiring SLAs
	2.3 Measuring SLAs
	2.4 Issuing audit reports
	2.5 Resource management and utilization
	2.6 Proactive monitoring of thresholds
	2.7 Predictive impact analysis
	2.8 Historical impact analysis
	2.9 Lifecycle impact analysis
	2.10 Static impact analysis
BPM Score (BPM)	1.1 Processes are easy to change
	1.2 Externalized business rules
	1.3 Runtime rules housed in a rules engine
	1.4 Use of process orchestration engine
	1.5 Monitoring of key performance indicators
	1.6 Use of policies
Loose Coupling Score (LC)	1.1 Service realization method
	1.2 Service access method
Governance Score (Gov)	1.1 Skilled Enterprise architect
	1.2 Skilled App/SOA architect
	1.3 Skilled project manager role
	1.4 Governance documented
	1.5 Governance understood
	1.5 Governance applied
	1.6 Services in repository / easy to discover
	1.7 Requirements changes are controlled
	1.8 Process defined for sunsetting services

The identified factors were then grouped according to the dimensions reported in Table I. Although not reported in this paper due to space constraints, we later conducted a statistical factor analysis which analyzed each of the 80 identified factors to determine the extent to which it correlated with the claimed business agility of a project [6]. As a result of this analysis, we identified the 36 factors depicted in Table II, as the practices exhibiting the most significant correlation with

business agility. In the remainder of this paper, we report results against these 36 factors only.

For each business agility factor identified in Table II, corresponding questions were developed for the survey. For example, two of the architecture questions were worded as follows:

- Did you use an enterprise service bus as part of your architecture to be the generic address for the majority of services? (i.e. You did not invoke services directly through point to point connections.) (Yes/No)
- An Atomic Service is a web service that can stand on its own and does not require the use of other web services to complete its functionality. In contrast, Composite Services are composed of other web services through aggregation, also referred to as structural composition, or at runtime through invocation and orchestration of services through use of a workflow engine. What is the percentage of services that are atomic for this project? ( )More than 90%,( )71-90%, ( )51-70%, ( )31-50%, ( )11-30%, ( )1-10% .

Results from the survey were then tagged and analyzed. Every factor was given a score that aggregated the answers to questions that are specific to that factor. As different questions were asked in different ways, the answers needed to be encoded. For example, 'Yes/No' questions were encoded as 0 and 1 respectively, while questions with answers on an ordinal scale were assigned an appropriate number of points within the range from 0 to 1.

3 SOA Projects

Data was initially collected from 40 projects; however three projects were rejected because they did not meet the inclusion criteria, while three were rejected due to significant amounts of missing data that made their collected data questionable. A second round of data collection started in late January 2011 and lasted for one month. A total of 9 non-IBM projects were used for the data collection process, and participants answered only the 80 questions related to our study. As a result, 39 projects were ultimately included in the data analysis. Of these, 30 projects were executed by IBM professionals, while 9 were non-IBM projects. In both cases the majority of project participants were from the US.

Categorized by industry, 21% of the projects were from Government, 12% from banking, 9% each from Financial Services, Telecommunications, and Healthcare, 6% from Insurance, and the remainder from other industries. 9% of the projects reported completion within three months. 27% reported durations of three months to one year, 21% took between 1 and 2 years, and the remaining 43% took over two years.

In all cases, the study participants were the tech leads and/or the architects that presided over the project during SOA solution implementation. The majority of the architects who filled out the surveys can be considered experts in SOA. Participants were given about two months to complete the survey due to the large number of questions and the level of required details. On average, most participants took about three hours to fill out the survey and additional time to collect the required data.

As part of the survey process, the project leads were asked to directly assess whether their SOA deployment contributed positively to their organization's business agility. Their response was recorded as a simple Yes/No answer. 64% of projects (i.e. 21 projects) were classified as contributing to business agility, while the remaining 36% (12 projects) were classified as non-business agile. The categorization of projects was totally independent from the other factors that were claimed to contribute to business agility.

4 Business Agility Factors

In this section we provide a more detailed discussion of the SOA claims that different dimensions of practice contribute to achieved business agility. For each of these areas we then report the survey results and discuss the implications of the results upon business agility.

4.1 SOA Architecture

Claim: A typical SOA system is expected to include certain components such as an enterprise service bus[5][3]. The architecture score is calculated through a set of attributes that all have equal weights. As depicted in Table II, the SOA score tracks architectural deliverables, monitoring decisions and certain data quality attributes, use of ESBs, etc.

Rationale: Poorly architected systems will not fulfill their potential in meeting stakeholder's objectives [7]. Given that there is no way to enforce SOA best practices [8] [3], a SOA score is required to measure how each project adheres to top best practices.

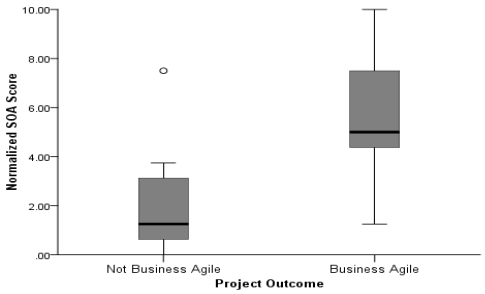


Fig. 1. Project's business agility outcomes and SOA Architecture score

Example: A good architecture that is attempting to enhance business agility should have provisions for handling events and alerts easily.

Survey Results: Fig. 1 shows the relationship between projects that achieved, or did not achieve business agility and

their corresponding SOA Architecture score. The SOA Architecture score is normalized on a scale from 0 through 10, with 0 meaning that few architectural practices were observed, and 10 meaning that all identified architectural practices were observed. The results show a fairly strong correlation between SOA architecture and business agility, with projects claiming business agility primarily scoring above 5, while those that did not claim business agility had generally lower scores.

An independent-samples t-test was conducted to compare SOA scores for business agile and non-business agile projects. There was a significant difference in SOA scores in business agile projects (M=5.58, SD=2.34) and SOA scores for non-business agile projects (M=2.08, SD=2.15); t(30)= -4.33, p = 0.000. The results suggest that SOA architecture factors are significant in predicting business agility outcomes of projects. Specifically, the results indicate that as projects adhere more to SOA architecture guidelines, the better business agility outcome of the project.

4.2 Business Process Management (BPM)

Claim: According to the Association of Business Process Management Professionals (ABPMP), BPM is defined as: "a disciplined approach to identify, design, execute, document, monitor, control, and measure both automated and non-automated business processes to achieve consistent, targeted results consistent with an organization's strategic goals" [9]. ABPMP emphasizes the role of collaboration, technology-driven, improvement and streamlining and the end-to-end management of business process to meet business goals and objective with "more agility." It is generally believed that using BPM components and solutions in combination with SOA enables an agile enterprise and facilitates business agility.

Rationale: The optimization of business processes as part of

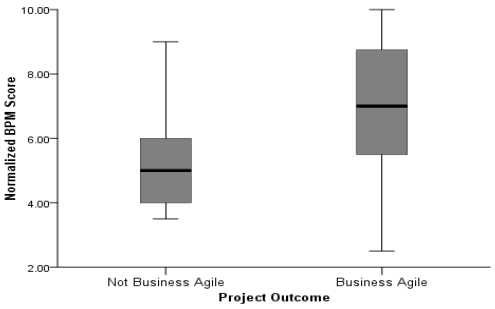


Fig. 2. Project's business agility and BPM Score.

building SOA solutions is one of the key factors for realizing business agility requirements [11]. The use of modeling and simulation in SOA helps drive the discovery of additional requirements especially those that may impact business agility. It can also uncover bottlenecks of the business process and expose additional areas that are ripe for streamlining.

Example: The level of business process modeling that is accomplished for a project, level of simulation used, level of Key Performance Indicators monitoring, and the level of use externalized business rules and service implementations.

Survey Results: Fig. 2 shows that projects that achieved business agility were likely to score higher on the BPM score. The majority of projects that received higher BPM scores reported achieving some business agility benefits for their projects. Projects that did not claim business agility benefits tended to score lower on the BPM scale. An independent-samples t-test was conducted to compare BPM scores for business agile and non-business agile projects. There was a significant difference in BPM scores in business agile projects (M=6.97, SD=2.04) and BPM scores for non-business agile projects (M=5.25, SD=1.6); t(28)= -2.65, p = 0.013. The results suggest that BPM factors are significant in predicting business agility outcomes of projects. Specifically, the results indicate that as projects adhere more to BPM guidelines, the better business agility outcome of the project.

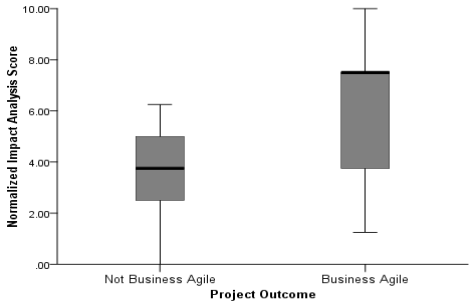


Fig. 3. Project's business agility and Impact Analysis Score

4.3 Impact Analysis

Claim: According to Bohner et al [10], change impact analysis involves "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change". Identifying changes and assessing their impact is a key requirement for achieving business agility. In SOA, Key Performance Indicators (KPIs) represent the strategic quantified goals of an organization and its services. KPIs provide specific performance objectives, often stated in terms of response times, throughput, latency, security, reliability, usability, accuracy, or cost. As such, KPIs can be used to evaluate whether a deployed system is currently achieving its stated business goals. The concept of service level agreements (SLAs) also plays a role in the implementation of an overall strategy for change impact analysis in SOA solutions. Therefore, observed runtime deviances from advertised KPIs and SLAs triggers change impact analysis steps.

Rationale: Business agility's demand for proactive sensing mandates that underlying systems are equipped with the proper structures that can sense changes immediately and provide corrective measures.

Examples: The ability to measure SLAs and assign thresholds in order to take corrective measures proactively. Services might not deliver on its previously defined SLAs (e.g., performance, availability, or cost), and a decision must be made as to what to do with underperforming services.

Survey Results: The data for Impact Analysis shown in Fig. 3, shows that projects which achieved business agility were likely to score higher on the Impact Analysis score than those that did not achieve business agility. An independent-samples t-test was conducted to compare Impact Analysis scores for business agile and non-business agile projects. There was a significant difference in Impact Analysis scores in business agile projects (M=5.93, SD=2.68) and Impact Analysis scores for non-business agile projects (M=3.75, SD=1.68); t(30)= -2.8, p = 0.008. The results suggest that impact analysis factors are significant in predicting business agility outcomes of projects. Specifically, the results indicate that projects which adhere more closely to impact analysis guidelines tend to achieve better business agility outcomes.

4.4 Governance

Claim: According to Brown [11], "SOA Governance is an extension of IT Governance that is focused on the business and IT lifecycle of services to ensure business value". Properly governed SOA services are those services that are funded properly for an obvious business reason and tie

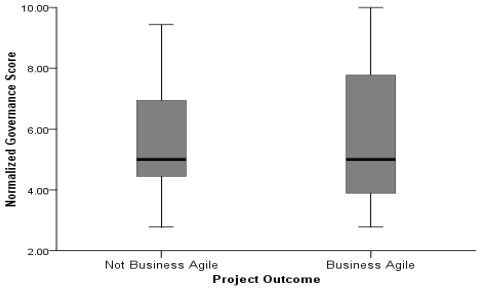


Fig. 4. Project's business agility and Governance Score

directly to business goals and objectives. Moreover, governed services are properly advertised, managed, secured and deployed properly to an infrastructure that will meet execution demands. Therefore, ensuring that services are well-governed is a key attribute of well-built SOA solutions [11], [3].

Rationale: Agile enterprises are efficient and control their resources. Therefore, better business agility can be achieved through governance of services including their proactive monitoring.

Examples: Existence of an enterprise architect and project manager. Tracking of requirements changes. Rules for managing and creating services.

Survey Results: Surprisingly, the results from our survey, depicted in Fig. 4, provided only weak support for the claim that better governance practices led to an increase in business agility. We found that projects which achieved business agility tended to include a wide degree of governance practices, with some projects claiming business agility scoring quite high, and some scoring quite low on the governance scale. These initial observations were supported by the fact that independent-sample t-test results did not indicate significance for this factor. These results warrant further investigation to determine if other factors such as the size of the project impact the importance of governance within a project.

4.5 Loose Coupling

Claim: Coupling refers to the degree of interconnections between software solution modules [12]. According to [13], loosely coupled systems are easier to maintain and understand,

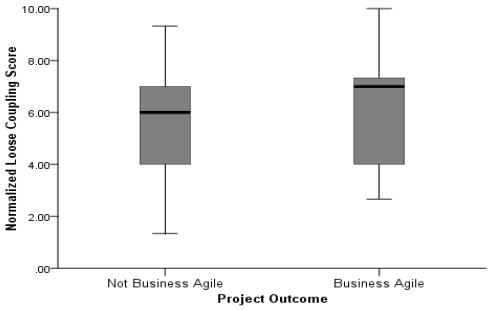


Fig. 5. Project's business agility and Loose Coupling score

as changes can be made independently to different modules without significant ripple effects.

Rationale: Loosely coupled architectures facilitate faster change with little impact to other solution components. Business agility is grounded in faster and more efficient handling of events and requires underlying structures to be capable of quick change with minimal or no impact to existing systems. Fiammante [3] points to some issues that plagued some of the early adopters of SOA and BPM due to their lack of experience in using the right tools and making the right architectural decisions. For example, some companies that adopted SOA failed to realize the power of loose coupling among business process components and the underlying services. This in essence led to the creation of rigid business process that largely failed to achieve the desired business agility benefits.

Examples: There are many service realization patterns that can be used for exposing and using services including the two primary patterns of Direct Exposure (DE) and Access Services. DE refers to exposing current IT systems or modules as a service without having to go through an intermediary component. Access services, on the other hand, refer to exposing current IT systems or a module as a service by going through an intermediary component such as an EJB. Access services are more loosely coupled than direct exposure services. Moreover, the use of virtualization layers, or

enterprise service bus, injects additional loose coupling into the overall structure of a SOA solution.

Survey Results: Fig 5 shows that projects which reported achieving business agility benefits tended to score higher on the Loose Coupling scale. Furthermore, the concentration of projects that achieved business agility reported scores of Loose Coupling that were closer to the mean. This indicates that some form of loose coupling is probably sufficient to enhance the outcome of business agility. Extreme loose coupling may have the potential to increase the complexity of the overall solution. In fact, SOA best practices advocate the need for building ESBs as major components of any SOA solution [8]. Independent-sample t-test results did not indicate significance for this factor. However, the data trends for business agile projects do suggest that business agile projects scored slightly better when comparing the means of business agile and non-business agile projects.

5 Analysis of Results

As summarized in Table III and in Figure 6, a closer investigation of the factors and their meanings after being normalized on a scale from 0 through 10 reveals that projects that claimed contributions to business agility had mean value for SOA score 173.08% higher than those projects that did not claim achieving business agility. A less significant result was reported for BPM, Impact Analysis scores and Loose Coupling score with values 33.76%, 58.13% and 9.49% respectively. As for Governance there was a minor difference (1.93%) in the mean between projects that achieved business agility versus those that did not.

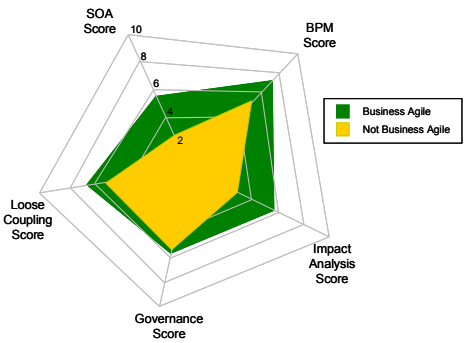


Fig. 6. Means for factor scores for project outcomes

TABLE III
MEANS OF PROJECT OUTCOMES

	SOA	BPM	IA	Gov	LC
Business Agile Mean	5.68	6.97	5.93	5.80	6.23
Not Business Agile Mean	2.08	5.25	3.75	5.69	5.69
Change Percentage	173.08	32.76	58.13	1.93	9.49

The significance of the SOA score is expected. Reaching the right architectural decisions and having the right skilled roles within a SOA project has always been a commonly understood approach for achieving business agility as well. The lack of divergence among projects with respect to Governance shows that while Governance is important in general, having significant amount of Governance is not a predictor of business agility outcome. Further detailed statistical analysis is required to determine the level of significance of Governance as a contributor to business agility. The other factors such as BPM score, Loose Coupling and Impact Analysis all showed some divergence between the means of business agile, versus non-business agile projects, indicating that such factors play a role in achieving business agility in SOA projects.

Fig. 7 illustrates these results for three specific projects, all of which claimed to have achieved business agility. As depicted in the graph, all three projects achieved higher scores in SOA, BPM and Loose Coupling. Project 1 achieved a score of

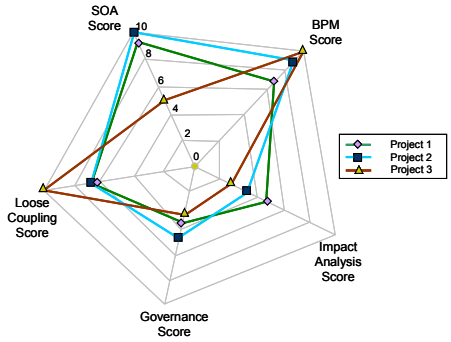


Fig. 7. Normalized score for business agile projects

(8.75) and Loose Coupling score of (7.33), both higher than the mean, while BPM and Impact Analysis scores were close to the mean. The Governance score (3.89) was 52% lower than the mean. This is another indication that SOA scores have a much greater impact than the other factors, and that BPM and Loose Coupling are also important. Projects 2 and 3 show a similar pattern. The graph indicates that each project exhibited relatively high scores for at least two of the top three factors of SOA Score, BPM and Loose Coupling.

6 Evaluation Process

The identified factors can be used to evaluate the potential business agility of a project. The process involves the following steps and is illustrated for one of the projects in our study:

1. Compute the SOA, BPM, Impact Analysis, Loose Coupling, and Governance scores.
2. Plot the results onto the spider diagram.
3. Compare the results against the means for business agile and non-business agile projects as reported in Figure 6.
4. Conduct a gap analysis by identifying specific practices that are lacking for any low scoring factors.

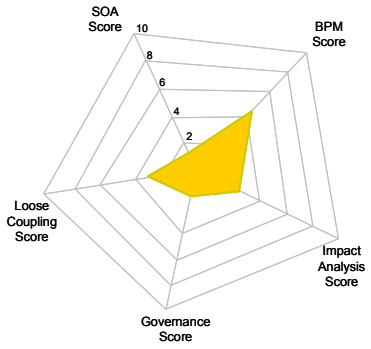


Fig. 8. Normalized score for a not-business agile project

TABLE IV
EVALUATED PROJECT DATA

	SOA	BPM	IA	Gov	LC
Evaludated Project Scores	1.25	4.50	3.75	2.78	3.33

TABLE V
EVALUATED PROJECT DATA SCORES COMPARED TO MEANS OF BUSINESS AGILE AND NOT BUSINESS AGILE PROJECTS

	SOA	BPM	IA	Gov	LC
Evaluated Project Score vs. Business Agile Mean Percentage Difference	354.40	54.89	58.13	108.80	86.90
Evaluated Project Score vs. Non-Business Agile Mean Percentage Difference	66.40	16.67	0.00	104.84	70.70

5. Define and institute appropriate improvement initiatives.

In our case study, the computed scores are shown in Table IV, and the results are plotted in Fig. 8. The results show that the evaluated project scored below the mean across four factors; SOA, BPM, Governance and Loose Coupling. For the fifth factor, Impact Analysis, the evaluated project had a score that was equal to the sample mean as shown in Table V. Gap analysis revealed that both the SOA and Loose Coupling scores were particularly low. Further inspection of the data revealed that the project did not have an ESB layer and failed to achieve loose coupling. Moreover, the project had no support for managing and monitoring events and alerts, and therefore failed to provide adequate support for impact analysis. These observations can lead to remediation steps, which if executed prior to rollout of project can help achieve thee business agility objectives at a lower cost than proceeding without evaluation.

7 Conclusion

This paper analyzed several factors that are widely believed to support business agility in SOA projects. The early analysis revealed that important factors such as SOA score, Impact Analysis, BPM score and Loose Coupling are important factors to be considered when aiming at achieving business agility. On the other hand, Governance score while still relevant was not as significant compared to the other factors.

The results from our study can be used to help project managers evaluate their projects, predict whether their projects are likely to achieve business agility, and recommend remedial actions that can improve potential business agility.

One of the key challenges for dealing with business agility is the lack of measurement tools to assess the level of attained business agility. Future work will focus on creating a measurement tool for assessing business agility. The measurement tool is important since it can be used in addition to experts' opinion for validation purposes. Once a measurement tool is identified and validated, the identified indicators can be further examined to construct a model that can predict the attainment of business agility as a result of SOA solutions. It is worth noting that attaining business agility as a result of SOA solutions is not a binary outcome. There are many degrees of attaining business agility and the identified factors need to be examined while taking this fact into consideration.

8 References

[1] R. Dove, Response ability : the language, structure, and culture of the agile enterprise. New York: J. Wiley, 2001.

[2] D. McCoy and D. Plummer, "Defining, Cultivating and Measuring Enterprise Agility." [Online]. Available: http://www.gartner.com/DisplayDocument?doc_cd=139734. [Accessed: 10-Apr-2011].

[3] M. Fiammante, Dynamic SOA and BPM : best practices for business process management and SOA agility. Upper Saddle River NJ: IBM Press/Pearson, 2010.

[4] P. Kidd, Agile manufacturing : forging new frontiers. Wokingham England ;;Reading Mass.: Addison-Wesley, 1994.

[5] F. Cummins, Building the agile enterprise : with SOA, BPM and MBM. Amsterdam ;;Boston: MK/OMG Press/Elsevier, 2009.

[6] M. Hirzalla, P. Bahrs, J. Cleland-Huang, C. Miller, and R. High, "A Predictor Model for Evaluating Business Agility in Deployments of Service Oriented Architecture," presented at the Submitted to the European Software Engineering Conference April, 2011, Szeged, Hungary, 2011.

[7] L. Bass, P. Clements, and R. Kazman, Software Architecture in Practice. Addison-Wesley, 1998.

[8] P. Bahrs, M. Hirzalla, R. High, and S. Pappe, 2010 3rd SOA Case Study Best Practices Conference - with focus on business agility, cloud computing, smarter planet and BPM. IBM Academy of Technology, 2011.

[9] "ABPMP - The Association of Business Process Management Professionals," http://www.abpmp.org/. [Online]. Available: http://www.abpmp.org/. [Accessed: 18-Feb-2010].

[10] S. A. Bohner and R. S. Arnold, Software Change Impact Analysis. Los Alamitos: IEEE Computer Society Press, 1996.

[11] W. Brown, SOA governance : achieving and sustaining business and IT agility. Upper Saddle River NJ: IBM Press/Pearson plc, 2009.

[12] R. Pressman, Software engineering : a practitioner's approach, 7th ed. New York: McGraw-Hill Higher Education, 2010.

[13] W. P. Stevens, G. J. Myers, and L. Constantine, "Structured Design," IBM Systems Journal, vol. 38, no. 2/3, pp. 231-256, 1999.

Toward Security Analysis of Service Oriented Software Architecture

Hassan Reza, and Washington Helps
School of Aerospace Sciences
University of North Dakota,
Grand Forks, ND 58202 USA

reza@aero.und.edu

Abstract

This paper presents an analysis of security aspect of Web-Based applications that utilize Service Oriented Architecture (SOA). The architectural solutions which address security requirements are examined and compared with other quality attributes relevant to web-based systems. More specifically, a trade off analysis in which security is the main focus is performed to select an architecture that best meets security requirement.

Keywords: *Software Architecture, Service Oriented Software Architecture, Security, Quality Oriented Software Architecture, ATAM*

1. Introduction

The success or failure of any nontrivial system depends heavily on its overall representation commonly known as Software Architecture [2]. Software architecture of a system has shown to be very important in the realization of system wide qualities such as security, performance, etc. [7]. Without appropriate software architecture, it will be very difficult to ensure that the end product will meet customers need both in terms of functionality (what the system does) and quality (how the system does it). In the simplest form, software architecture of a system refers to the principal design decisions that include components (computational elements), connectors (interactions), configurations (overall structure), and constraints (rules) [2].

The advent of the Internet resulted in what is known as online business revolution. With almost every conceivable company and organization having a website on the net, it is now possible that websites to collaborate with each other to accomplish certain tasks or services that are very essential to overall customer transactions and satisfactions. For example, an online airline reservation system may need to collaborate with other website such as payment service provider PayPal to enable customers to buy their tickets and complete their transactions. Service Oriented Architecture (SOA) a collection of design principle used during software and system development lifecycle. More precisely, it is a software architecture that can be used to implement business services a set of loosely coupled, autonomous and distributed components aimed at delivering a well defined level of service [3,8]. At the heart of SOA is the notion of services or tasks (e.g., providing credit card information). Solutions to the customer requests are created by interfacing and integrating dispersed services using XML and SOAP (Simple Object Access Protocol) respectively.

The architectural style of SOA, for most part, shares a lot of similarities with client/server (or call/return) architectural style. SOA begins with a service consumer sending a request to a service provider. The service provider then processes the request by performing a set of actions or services. It then returns the result to the requester. The way that SOA works is slightly different from the traditional client/server style because in SOA, servers can play duel role of clients and servers. Moreover, SOA provides higher degree of flexibility, reusability, and interoperability [4].

An SOA is comprised of three participants and three fundamental operations, regardless of its implementation [5] (see figure 1). The three SOA participants interact through three basic operations: publish, find, and bind.

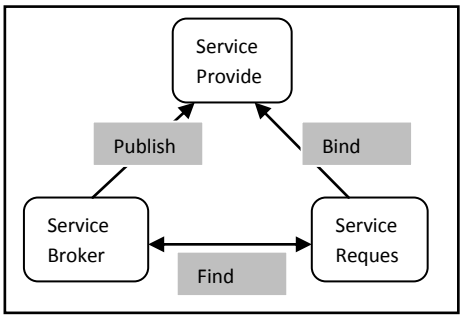


Figure 1: The SOA model according to IBM

The security is a major issue in web-based systems and SOA. It is blamed for the failure of web-

based applications regardless of what standards and protocols (e.g., SOAP and/or WSDL) have been enforced [6,21]. Other non-functional requirements (NFR or Quality Attributes) relevant to web-based systems and SOA are performance, usability, modifiability and availability and inter-operability. Performance is a quality attribute with well-established area of its own research works. Performance refers to the degree of efficiency and effectiveness by which the system responds to external stimulus. Usability refers to the ease by which users can interact with the system in order to complete a task. Modifiability is an attribute which describes the ease by which a system can be modified to add, delete, or update a feature system [7].

As discussed in [9, 12], SOA presents a new set of challenges because components are loosely coupled and are exposed as independent services on a network. Developers are blamed to fail close attention to the security aspect of SOA services. As such, vulnerabilities issues such as injection flaws, issues related to XML Denial of Service, insecure communication, and insufficient authentication are magnified in SOA.

2. The Security Requirements

Security is a very broad issue and it means different concepts to different domains and/or entities. In general, security refers to the ability of a system to withstand attacks and unauthorized accesses. Security is a collective term and therefore consists of a set of sub-attributes. The sub-attributes associated with security include: 1) confidentiality, which refers to authorized access to information/services; 2) trust, which includes both authentication and authorization; 3) integrity, which ensures that information is not corrupted; and 4) availability, which ensures that the service is available whenever it is requested. Because security is a very broad concept, we attempt to limit the scope of security to those issues related to web-based systems. The most significant of these concerns include access control, communication security, and availability.

2.1 Identity requirement

In a SOA environment, one of the design decisions for security must include identity that must be decoupled from the services. In a SOA, both users and services have identities. These identities need to be properly distinguished so that appropriate security controls can be applied [10]. Furthermore, the identities may need to be propagated throughout the SOA environment. Propagation is the scenario in which a user or service may need to access multiple layers of services. Therefore, identity services are required in the infrastructure to deal with identity mediation issues in a way that services can be easily interconnected without any concern about how to map and propagate user identity from one service to the next.

As discussed in [11], the authors made a point that managing identity authentication, authorization, accounting, and security is often neglected in SOA. The sheer of complexities involved in managing identity in a distributed network requires some mechanism to deal with issue. Federated identity management (FIM) is one such mechanism and it can be defined as an industry framework built on top of industry standards. FIM allows subscribers from different organizations/entities use their internal identification data to obtain access to the networks of all enterprises in the group.

2.2 Authentication and Authorization requirements

Authentication is a mechanism that provides the system with the ability to verify the authenticity of an identity. Authentication does not only include mechanisms such as user name and password, but can also support hardware tokens such as Media Access Control (MAC) addresses and biometric data. A typical authentication process involves collecting the consumer's identity and authentication credentials and evaluating that the credentials presented by the consumer correspond to credentials that are expected to be presented by the user. Authentication Services together with identity propagation described in the previous section provide solution components that enable end-to-end identity flow which is considered as one of the most significant security challenges in SOA [10].

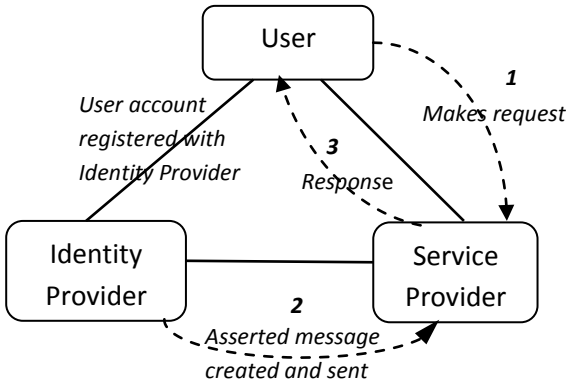


Figure 2. Basic SAML structure and operation

A known method of providing authentication for web-based applications is SAML (figure 2). SAML is a secure XML-based communication mechanism for exchanging authentication and/or authorization of data between entities playing the role of an identity provider (a producer of assertions) and a service provider (a consumer of assertions). SAML eliminates the need to have multiple authentication credentials such as passwords in multiple locations. This is very important, because such a mechanism increases security by eliminating additional

credentials which in turn may reduce the opportunities for identity theft and phishing.

Authorization follows authentication which means that once a user has been authenticated, authorization can then be performed. In general authorization has to be flexible and supports various techniques such as role based or attribute based access control. It also should be independent of the authentication in order to provide modularity. [14]

2.3 Confidentiality and Integrity requirements

Confidentiality relies heavily on cryptographic techniques such as encryption. This usually means encrypting the data so that it cannot be read by unauthorized persons and only someone in possession of the decryption key can read the data.

Secure Sockets Layer (SSL) is an example of a secure transport level scheme. However, the WS-security specification has been found to be more effective in providing message-level security because it can provide end-to-end message level security. This feature, in turn, allows the messages are protected even if the messages may go through multiple services, or intermediaries [10].

WS-Security is very flexible and is designed to be used as the basis for securing Web services within a wide variety of security models including Public Key Infrastructure (PKI), Kerberos, and SSL. Other characteristics of WS-Security are support for multiple security token formats, multiple trust domains, multiple signature formats, and multiple encryption technologies to provide integrity or confidentiality.

2.4 Availability

Availability of a service or resource implies that the system is able to provide a response to a request in a timely manner. Architecting for high availability includes application clustering, database clusters, and similar techniques. Availability can be improved if a service provider can build into its applications contingencies plans such as exception and error handling mechanisms whenever a service becomes unavailable [15].

3. Related Work

Security has various effects on other quality attributes important in web-based systems. As mentioned in previous section, quality attributes (QAs), such as performance, security, modifiability, reliability, and usability, have a significant influence on the software architecture of a system [17] and hence on the quality of web-based systems (or ecommerce). In what follows, we discuss related works relevant to security mechanisms and their impacts on other qualities attributes.

3.1 Trust Mechanisms

FIM positively impacts usability since users do not have to provide credentials multiple times (single-sign-on). Scalability is positively impacted as well because any number of users can be added or removed or modified easily. Reusability and modifiability are other QAs that are positively impacted by FIM. On the other hand, performance as an QA may be negatively impacted because validating an identity has to be done through another service which can be separated from the requested service. This, in turn, may cause overhead stemming from additional message passing.

For authentication purpose, the solution must ensure that a consumer that has already been authenticated is accepted by the service provider as being authenticated. SAML facilitates this by binding security tokens to the request.

Another consideration is how rigid the authentication policy must be. This is important from an architectural point of view because stronger authentication requirements (i.e., adding multiple levels of authentication) tends to negatively affect a number of QAs including performance (overhead from authentication calls and increased message size), and usability (complexity in managing certificates and tokens and the difficulty for users to create stronger passwords).

3.2 Message Security Mechanisms

For message security, WS-Security is the standard that is enforced in a SOA environment. WS-Security allows for end-to-end message level security. In addition to this, security can be embedded as part of the message. However, the downside is that complexity will be increased because it requires careful management of parts needed to be secured. For message security, the design must be the one that provides a mechanism which mitigates against a man-in-the middle security breach.

Architecting with the mechanisms for availability as a security requirement will have a negative effect on modifiability since the complexity involved in clustering and exception handling (the dynamism involved) will be significant. In general, availability as a security requirement has a very positive impact on usability because highly available system makes the system more usable.

3.3 Security Tradeoffs

Security as a quality attribute is critical in web-based systems, but it is also one for which there are tradeoffs with other quality attributes in the system under construction [17]. Security is a before-fact and hence must be considered early on in the development process. Therefore, it is important that a tradeoff analysis be performed early on in order to understand the impact of security on the overall software quality and consequently select an

architecture that best meets the security requirement without distributing other QAs. More specifically, the optimal architecture is the software architecture that should not only satisfy the security requirement of a SOA but will also make reasonable compromises that minimizes the negative impacts of security on the other desired quality attributes such as performance and usability which are relevant to web-based systems. To this end, we documented the relative impacts of security on other QAs. See Table 1 which shows, security impacts performance negatively.

For example (see Table 1), propagating a consumer identity requires extra message, which in turn may require extra message calls to validate the identity and to append security tokens to the message. Furthermore, if one were to consider making even such identity validation more secure by encrypting the message. Encrypting the message, in turn, will add additional overhead because the process requires additional resources such as CPU time to encrypt and decrypt, and network bandwidth.

QAs of SOA	Impact on Security
Performance	-
Usability	-
Modifiability	-
Availability	+
Interoperability	+
Reliability	++
Scalability	++

Table 1. Impact of Security and QAs: -:negative; +:positive; ++:very positive; --: very negative.

Another scenario which shows how security negatively impacts performance involves the requirement for message level security. For instance, the integrity of messages must be maintained to ensure that unauthorized changes are not made to them. That is, the data must be delivered as it was intended. In order to enforce this requirement, some schemes involve encoding redundant information such as checksums and encrypting the entire message. Again, this results in similar overhead as described in the previous example and thus leads to the performance degradation such as increased latency.

Security negatively affects the usability of the system [7]. In an ecommerce application this is a very important quality attribute because there is a wide cross-section of users from novices to power users. An example of usability which involves an architectural aspect is the type and rigidity of the authentication requirement of users (i.e., the restrictions imposed on users to create more sophisticated passwords). Very rigid password requirements will make it difficult to use the system because users may have difficulties remembering them leading many users to writing them down or to reset their passwords constantly to new ones.

Therefore, careful attention must be paid to the level of security that is deemed to be needed in the context in which the system is being used. For usability, the impact depends on the degree of authentication which is required by the individual application, where the greater the degree, the greater the negative impact.

Security also negatively impacts modifiability as discussed in [17]. The mechanisms to address the security requirements of the system and how they are designed will play a significant role in determining how easy it will be to make changes to the system in the future.

Architecting with the mechanisms that support availability as a security requirement will have a negative effect on modifiability because the complexity involved in clustering and exception handling (the dynamism involved) will be significant. Availability as a QA strives for back-up and redundancy while security strives for minimality. It is very obvious negative correlation between these two requirements and how this correlation leads to a conflict. The mechanisms which improve the availability of a system such as clustering and back-ups effectively increase the security needs and the complexity. This complexity, in turn, amplifies the likelihood of a security breach occurring.

4. Optimizing SOA for Ecommerce

We have borrowed the notion of quality attribute scenarios and tactics initially developed by [18,19] to document nonfunctional requirements of a web-based systems. To this end, the quality attributes are expressed using a set of quality scenarios. The scenarios have not been utilized to drive an architectural style but rather they have been used to highlight the architectural patterns which may play a role in the optimization process. The final stages of this process constitute our selection process which include application of the architecture trade-off analysis method (ATAM) [18, 19]. Figure 3 shows an initial SOA model before applying the ATAM for the security QA.

Architecture is critical to achieving the desired qualities of a system such as performance, usability and security in the case of ecommerce systems [7, 13, 18, 19]. A scenario based approach ensures that the non-functionality of a typical web-based system which is built using SOA. At the heart of scenario-based approach is the creation of utility tree (UT) that can be used to characterize qualities attributes associated with a SOA and ecommerce systems. Other important concept is a design tactic. Tactic is a micro design solution used to implement a particular aspect of a given quality attributes [18].

As discussed in [8, 19], ATAM process consists of four main steps as follows: 1) scenario and requirements gathering, 2) architectural views, 3)model building, and 4) tradeoffs analysis.

Additional iterations of the above steps can then be carried out in order to re-evaluate the model to meet the NFRs of a system. The first two steps have already been covered in previous sections. For the purposes of this paper the focus will be on the attribute specific analysis, sensitivity and tradeoff identification of the security attribute.

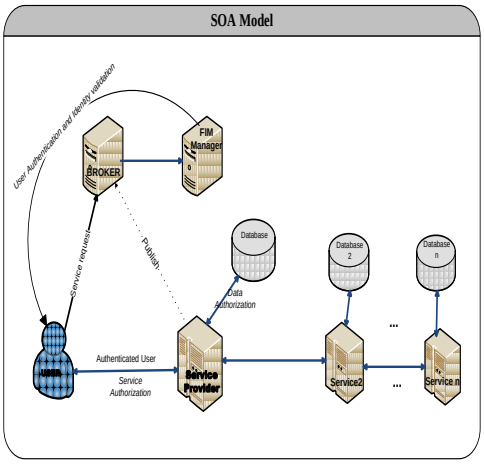


Figure 3. A typical SOA model

The system can resist attacks by employing Publish-subscribe, repositories, and FIM because all these patterns support the following tactics: breaking dependency chain, identity management, authorization, authentication, message security mechanisms or tactics. For instance, FIM can be used for maintaining trust and integrity which is very essential to SOA. FIM supports authentication, authorization, confidentiality and data integrity. FIM also enables end-to-end authentication which is suitable for usability but it may compromise security. With end-to-end authentication, a consumer after initial authentication does not need to re-authenticate regardless of the number of additional services (internal or external) which may be accessed in order to complete a task or transaction. For example, if on one of the hops a hacker intercepts a message, that hacker may be able to access a service because the authentication for the hacked user is assumed to be valid by 'downstream' services. Point-to-point security would eliminate this kind of security breach, but it would compromise usability (a user may have to re-authenticate multiple times in a single session) and this impact the unique benefits of SOA.

Another important consideration involves the availability requirement. For example, the scenario in which a service may go down, consumers will have to re-discover that service. This, in turn, eliminates the possibility of spoofing or service masquerading, because the consumer would have to be re-authenticated again. This may, however, affect reliability because if an order was being processed and payment had already made, recovery should not result in the consumer being billed twice for the same order for instance.

Caching is a popular tactic used in Client/Server architectures [1] to improve performance of a system; it can also be employed in a SOA setting to keep a record of the transaction state that is checked upon re-initiation with a service. Figure 4 depicts the same SOA model after we applied ATAM. In that figure, cashing has been incorporated into model as part of the service provider. In addition to address the above problem which includes security and reliability, caching could also reduce contention and demand for server (service) resources, thus improving both performance and scalability.

This brief analysis demonstrated how ATAM can be used to discover various tradeoff points. From this point, ATAM iteration will be required in order to re-evaluate and hence to incorporate the changes for those set of attributes which may have been affected by the security analysis. For example, the manner by which security is addressed when a service fails (requiring rediscovery and subsequent re-authentication) leads to the addition of caching as a pattern to improve reliability, performance and security. Thus performance and reliability need to be re-evaluated in order to incorporate this change

An ecommerce system based on SOA involves the connection of many sub-systems, external entities and technologies which may add additional complexity that must be considered when designing the system's architecture. This is why a method such as ATAM is good to use to provide evaluation from any desired quality attribute perspective. Such a method has shown for tradeoffs and sensitivities which can impact the system or other quality attributes in negative or positive ways.

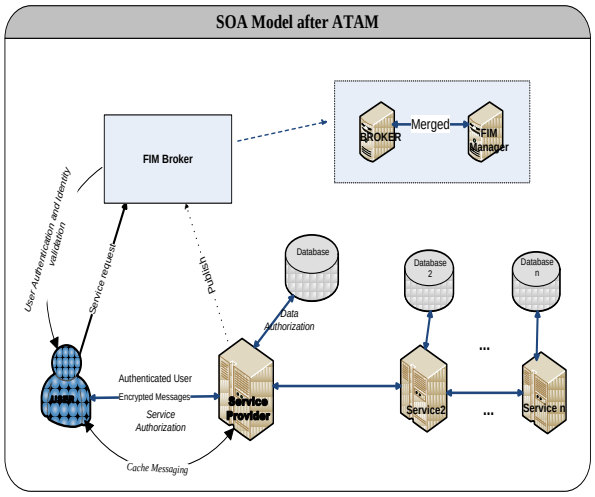


Figure 4. SOA model after ATAM is applied

The analytical framework provided by ATAM can help us to determine the characteristics of each of the architectural patterns and highlights their costs and benefits [19]. The ATAM also helps architects to determine architectural tradeoff points, which in turn increase our understanding because the approach highlights the limitations and liabilities of suggested

architectural solutions. The end result is the development of informed action plans for modifying the architecture, leading to new evaluations and possibly new enhancements.

The complexity which is intrinsic in most real-world software design means that an architecture tradeoff analysis will rarely be a clear-cut activity that allows engineers to proceed linearly to select a perfect architecture [19]. As it has been our experience in this work, applying the method answers some design questions and at the same time it poses other concerns.

5. Conclusions and Future Work

In this work, we discussed the security requirements that is a collective term and consists of sub-attributes (e.g., integrity) in ecommerce systems and SOA. To this end, we examined the mechanisms which implement security.

The nature of SOA dictates that after understanding the security requirements, one must accommodate security in the system's architecture in a concrete way such that security as a quality attribute is not isolated and hence describes its correlation with other non-functional requirements (NFRs). The effects each security mechanism may have on other QAs (e.g., availability) must be understood in order to select optimal architectural design that also meets other NFRs of web-based system utilizing SOA.

This work is by any means complete and therefore requires additional work. As such, it requires additional work focusing on qualitatively analyzing of security schemes to address security requirements so that their impact on the overall system quality is clearly defined and an optimal security scheme can be selected. In addition, the method employed in this paper must be experimented with other quality attributes in other domains to validate its feasibility in deriving optimized architectures.

References

[1] F. Buschman, R. Meunier, H.Rohnert, P. Sommerlad, M. Stal. A System of Patterns. Wiley 1996.

[2] R. N. Taylor, N. Medvidovic, and E. M. Dashofy; Software Architecture: Foundations, Theory, and Practice," John Wiley and Sons, 2009.

[3] J. Hurwitz, R. Bloor, C. Baroudi and M. Kaufman, "Service Oriented Architecture For Dummies," John Wiley & Sons, 2007.

[4] M. Matsumura. "Intentional SOA for Real-World SOA Builders," Software AG, May 2007.

[5] J. Bih, "Service Oriented Architecture (SOA) A New Paradigm to Implement Dynamic E-business Solutions," *Ubiquity* , 2006.

[6] Gib Trub, L. O. (2008, September). *Global report on SOA/Web services security initiatives*," Retrieved September 22, 2010, from http://www.ca.com/files/industryresearch/gmg_worldwide_soa_research_survey_08.pdf

[7] H. Reza and E. Grant, "Quality-Oriented Software Architecture," *International Conference on Information Technology* (ITCC), 2006.

[8] E. P. Taylor. Security in a Loosley Coupled SOA Environment:http://www.developer.com/design/article.php/3605836.

[9] National Security Agency. (n.d.). *Service Oriented Architecture Security Vulnerabilities Web Services*. Retrieved October 3, 2010, from www.nsa.gov/snac: http://www.nsa.gov/ia/_files/factsheets/SOA_security_vulnerabilities_web.pdf

[10] IBM, "Understanding SOA Security Design and Implementation," IBM, November 2007. Retrieved from http://www.redbooks.ibm.com/redpieces/abstracts/sg247310.html?Open&pdfbookmark, on November 20, 2010.

[11] J. Oltsik, (November 2006) www.enterprisestrategygroup.com. *Enterprise Strategy Group.*

[12] E. Pulier and H. Taylor, (2006, May 19), "Solutions to SOA Security," Retrieved October 1, 2010, from http://www.developer.com/design/article.php/10925_3607471_1/Solutions-to-SOA-Security.htm

[13] L. Bass, P. Clements, R. Kazman. Software Architecture in Practice Second Edtion, Addison Wesley, 2003.

[14] K. Soo Hoo, A. W. Sudbury and A. R. Jaquith, "Tangible ROI through Secure Software Engineering," *Secure Business Quarterly 1*, 2 (2001).

[15] J. Li and A. H. Karp, "Access Control for the Services Oriented Architecture," *ACM*, Fairfax, Virginia, 2007.

[16] P. Bianco, R. Kotermanski and P. Merson, Evaluating a Service-Oriented Architecture," *Technical Report - Software Architecture Technology Initiative*, Software Engineering Institute, Carnegie Melon University, September 2007.

[17] G. Boella, L. van der Torre, "Permission and Authorization in Normative Multiagent Systems" in *ICAIL '05 Proceedings of the 10th international conference on Artificial intelligence and law*, Bologna, Italy, June, 2005.

[18] L. Bass, P. Clements and R. Kazman. "*Software architecture in practice*," Pearson Education Inc., 2003.

[19] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson and J. Carriere, "The Architecture Tradeoff Analysis Method," *Software Engineering Institute*, Carnegie Melon University, July, 1998.

Efficient Aspect Assignment in Heterogeneous Distributed Systems

S. Bulu, and F. Buzluca
Department of Computer Engineering, Istanbul Technical University, Istanbul, Turkey

Abstract - *In recent years, with increasing use of distributed systems, distributed Aspect Oriented Programming (AOP) arouses more interest. The way of distributing aspects over the network can affect the performance of the program. While assigning aspects to hosts, to achieve higher performance, properties of the distributed system and relation between aspects and objects must be taken into consideration. In this context as a solution for the aspect assignment problem in distributed systems we propose two algorithms, namely a Genetic Algorithm (GA) and an A* algorithm. We evaluate the efficiency of these two algorithms by using a simulator on heterogeneous distributed systems, where hosts and communication lines have different properties. It is shown that the GA is more favorable than A* algorithm for larger systems with many nodes while for smaller systems A* may be preferable. The simulation results indicate that the properly assignment of aspects to hosts improves the performance of a distributed aspect oriented program.*

Keywords: Aspect Oriented Programming, Distributed Systems, Task Assignment, Genetic Algorithm, A* Algorithm.

1 Introduction

Aspect-oriented programming (AOP) [1] is a programming style that allows programmers to implement cross-cutting concerns like logging, tracing, profiling, policy enforcement, pooling, caching, authentication, authorization and transactional management in a modular way and then combine these concerns with a base program through a process called weaving. AOP aims at improving the quality of the software by decreasing the level of code scattering and code tangling known as primary symptoms of non-modularization.

There are lots of AOP implementations that have been widely used. Some of these implementations are AspectJ [2], AspectWerkz [3], JBoss-AOP [4], PostSharp [5] and Spring AOP [6]. AspectJ, which was proposed as an extension of the Java language for AOP, is the most prominent implementation.

In an AOP cross-cutting concerns are defined as a set of aspects. An aspect consists of method-like constructs called advice. An advice is used to define additional behavior at a set of well-defined points called join points in the program's execution. Join points are matched by a predicate called pointcut. AOP weaver maps various crosscutting elements to the object oriented constructs. For example, aspects map to classes where each data member and method in aspect become the members of the class. Pointcuts are intermediate elements that map to methods. Advice usually maps to one or more methods. The weaver inserts calls to these methods at potential locations matching the associated pointcut. During the execution of an AOP objects call methods of related aspects and mostly objects and related aspects operate on common data.

Distributed systems are computer systems, in which the processing elements are connected by a network. They have become increasingly popular in recent years because of their high speed and high reliability. In such area, a new paradigm called distributed AOP has recently appeared. In distributed AOP, aspects can be deployed in a set of hosts. Remote pointcuts [7], which are similar to traditional remote method calls, invoke the execution of advices in aspects on remote hosts. DjCutter [7], JAC [8], AWED [9], Damon [10] and ReflexD [11] are approaches that address distributed AOP.

Distributed systems allow programmers to divide applications into a number of tasks and execute concurrently on different hosts. This process obtains tremendous improvement in the performance when the task distribution and assignment are applied effectively. A similar problem also exists in distributed AOP: How to assign aspects to the hosts in the network so that the time required for program completion is minimized?

While assigning aspects of an AOP to hosts in a distributed system several properties of the physical system and the program must be taken into consideration. These properties are processing capabilities of hosts, parameters of communication links, amount of data shared between objects and aspects. The assignment of aspects is critical and affects the program completion time because when there is a call from an object to an aspect (assuming that the object and the aspect are assigned to different hosts), exchanging data between these object and aspect consumes time. This time depends on the amount of the data and the capacity of the link which is used during the data transfer.

In this paper we first formulate the aspect assignment problem in heterogeneous distributed systems by taking all necessary parameters into account and then propose two algorithms to solve this problem that occurs in distributed AOPs. The first algorithm is based on Genetic Algorithms (GA) [18], and the second one is an A* [19] based search technique. We also evaluate the efficiency of these algorithms for different systems and programs. We compare the increase in the performance of the AOP obtained by these algorithms with an algorithm that assigns aspects to host randomly. Simulation results indicate that the assigning aspects to hosts properly using the proposed algorithms can reduce the completion time of a distributed aspect oriented program almost by half compared to randomly assignment.

The rest of this paper is organized as follows. Section 2 mentions related work. Section 3 gives background information on the aspect assignment problem. Section 4 and 5 present the two algorithms for the problem. Section 6 compares the efficiency of the algorithms and discusses the simulation results. Finally, section 6 concludes the paper.

2 Related work

Task assignment problem in distributed systems, which is similar to our aspect assignment problem, is well-known to be NP-hard [12]. Many approaches to this problem have been identified up to now. However, most of this reported approaches yield suboptimal solutions because of the large state space. Task assignment approaches can be classified into three categories, namely, graph theoretic, integer programming and heuristic methods.

In graph theoretical approach, each task or/and host is represented by a node and the cost induced by the communication delay between them is represented by a weighted edge. The first attempt in graph based approach is done by Stone [13]. In Stone's work, a Max Flow/Min Cut Algorithm is utilized to find assignments which minimize total execution and communication costs.

The integer programming method formulates the model as an optimization problem and solves it via mathematical programming techniques. For example, Chu [14] formulates the problem into a nonlinear integer zero-one programming problem and then reduced it to a linear zero-one programming problem.

As the first two approaches attempt to obtain the optimal solutions, they search the whole space and therefore need a lot of time and memory. Heuristic methods [15][16][17], on the other hand, do not pursue the optimal solutions but provide sub-optimal fast and effective solutions. They use special parameters that affect the systems in indirect ways.

Although there are a large number of task assignment algorithms, none of them is interested in assignment of aspect. To assign aspect to processing nodes properly we take the following two structures into consideration. Firstly, the parameters of the distributed system such as processing capabilities of nodes and bandwidths of communication lines between them. Secondly, structure of the aspect oriented program expressed by the relations between aspects and objects such as reference counts, amount of transferred data between them.

3 Problem definition

In this study we first formulate the problem, how to assign aspects of an AOP to hosts in a heterogeneous distributed system to increase the performance of the program. The assignment problem for aspects in distributed systems can be defined as the assignment of k aspects $A = \{a_1, a_2, \ldots, a_k\}$ to n hosts, $H = \{h_1, h_2, \ldots, h_n\}$.

We define our distributed system in the form of heterogeneous in which the connected hosts have different processing capabilities. In the network X_{iq} denotes the execution cost of aspect a_i that is proportional to the execution time of the aspect when it is assigned to and executed on host h_q, $1 \le i \le k$, $1 \le q \le n$. Here we assume that each advice in the same aspect has the same load. This means that the execution time doesn't depend on which advice of an aspect is executed.

Each communication link in the network has different costs of delay, which can be represented by a delay matrix $D=\{D_{pq}\}$. D_{pq} denotes the communication cost between two hosts h_p and h_q, which arise because of the communication delay when an object located on h_p calls an aspect located on h_q. Further, $D_{pq}=D_{qp}$ and $D_{pp}=0$.

Another parameter that effects the performance of the AOP is the relation count defined as how many times each aspect instance will be called from each object. These values can be obtained from the AOP framework tools. For example, AspectJ Development Tools (AJDT) allows programmers to register a listener to obtain crosscutting relationship information whenever a project is built [21]. Let there are m objects, $O = \{o_1, o_2, \ldots, o_m\}$, then R_{ij} denotes aspect-object relation count between aspect a_i and object o_j. On the other hand, when there is a call from an object to an aspect, a communication cost is incurred because of exchanging data. So, let C_{ij} denotes the communication cost between aspect a_i and object o_j that is proportional to size of data transferred between a_i and o_j. All cost values (X_{iq}, D_{pq}, C_{ij}) are normalized by assigning one to the smallest positive value in each group.

In our study we assume that locations of objects are fixed and predetermined according to their specific jobs. We focus on distributing and assigning aspects, which are used by the objects. Therefore we don't consider the execution times of objects. So, let L_j denotes the host that object o_j is assigned to, $1 \le L_j \le n$.

All parameters described in this section can be derived explicitly from the distributed system. As an example, the parameters of a simple system which is made up of three hosts, five objects and four aspects are represented in tabular form in Figure 1.

D_{pq}	h_1	h_2	h_3
h_1	0	1	3
h_2	1	0	2
h_3	3	2	0

a. Host Communication Costs

X_{iq}	a_1	a_2	a_3	a_4
h_1	1	4	2	3
h_2	3	1	6	2
h_3	5	6	2	3

b. Aspect Execution Costs

C_{ij}	o_1	o_2	o_3	o_4	o_5
a_1	4	1	2	4	4
a_2	4	1	5	3	2
a_3	2	2	6	3	5
a_4	5	2	1	5	4

c. Aspect-Object Communication Costs

R_{ij}	o_1	o_2	o_3	o_4	o_5
a_1	16	0	11	14	9
a_2	17	3	9	10	5
a_3	2	9	0	9	16
a_4	14	1	14	4	9

d. Aspect-Object Relation Counts

$O(j)$	1	2	3	4	5
$h(L_j)$	1	2	2	3	1

e. Object Assignments (L_j)

Figure 1. An Example set of system and program parameters

The solution of the aspect assignment problem is a proper mapping of k aspects to n hosts that will minimize the running time of the aspect oriented program. To evaluate the efficiency of the assignment procedure we consider the host that is maximally loaded by the aspects. Here the load of a host is defined as a metric that is proportional to the total time consumed by the aspects located on this host during the execution of the program. As the hosts in a distributed system run parallel the host that needs the longest time to complete its aspects is taken into consideration, because it will determine the completion time of the whole AOP. The load metric of a host consists of two components. First one is the total running time of the aspects on this host and second one is data transfer time between these aspects and related objects. Let T is the set of aspects that are assigned to host q then the load on host q is:

$$Load_q = \sum_{\forall i \in T} \left(\sum_{j=1}^{m} R_{ij} X_{iq} + \sum_{j=1}^{m} R_{ij} C_{ij} D_{qL_j} \right) \qquad (1)$$

where m is the number of the objects and L_j is the host number of j^{th} object.

The solution has to fulfill two objectives. First, we try to minimize the load of the maximally loaded host, which is represented by the following cost function $F1$:

$$F1 = \max(Load_q), \quad 1 \le q \le n \qquad (2)$$

This first objective is related to the completion time of the aspect oriented program under assumption that all hosts operate parallel. Secondly, if there are many aspect assignment possibilities, which minimize the $F1$, the second objective is to minimize the sum of load on all nodes, which is expressed by the following function $F2$:

$$F2 = \sum_{q=1}^{n} Load_q \qquad (3)$$

4 Genetic algorithm

Genetic algorithms (GA) [18], which are used for solving many search and optimization problems, generate solutions using techniques inspired by natural evolution. A GA starts by generating a random population of solutions (called chromosomes in GAs literature). At each iteration a number of solutions are selected for the mating pool according to their fitness. Crossover and mutation operations are then applied to mating pool in order to produce new solutions. The algorithm terminates when either a maximum number of generations has been produced or population is converged.

The first step in designing a GA is to develop a suitable representation for chromosomes in the population. In our algorithm we use integer representation, with considering the relationship between hosts and aspects. For k aspects there are k elements (called gene in GAs literature) in the chromosome. The value of each gene in the chromosome represents the host to which that aspect is allocated. As an example a chromosome with four genes is shown in Figure 2.

Aspect:	1	2	3	4
Host (gene):	2	3	1	2

Figure 2. Chromosome representation

The fitness value of each gene in the chromosome is the load on the host that the gene represents ($Load_q$), which is calculated using the equation giving in (1). On the other hand the fitness value of the chromosome is the maximum gene fitness which is the load on the heaviest-loaded host that is represented by $F1$ as given in (2).

In our algorithm, we apply one point crossover operation on a pair of chromosomes which is randomly selected from the mating pool. One point crossover is accomplished by randomly choosing a point along the length of the chromosome, and exchanging all genes beyond that point in either chromosome. This operation yields two new chromosomes. After crossover operation, a mutation operation is performed on a randomly selected gene of each chromosome with a certain probability. In mutation operation the value of a gene is replaced by randomly generated host number.

After crossover and mutation operations the worst chromosomes, the chromosomes with the highest value of fitness (*F1*) in the population, are replaced by new ones in the mating pool. This means that the best chromosomes (the chromosomes with the lowest value of fitness) in the population are carried to the next generation (called elitism in GAs literature). In our algorithm we replace 1 chromosome by new ones with better fitness values. If there are many chromosomes with the same fitness value (*F1*), then the second objective (*F2*) comes into play, and the chromosomes with the smallest *F2* value are selected. The complete GA is given in Figure 3.

```
Generate initial population (chromosomes represent
different aspect assignment possibilities, Fitness=F1)
do {
  Create mating pool
  Apply crossover operation
  Apply mutation operation
  Apply elitism (Select chromosomes with smallest F1
  values. If these values are equal select chromosomes
  with smallest F2 values.)
  Carry new chromosomes from mating pool to population
}until(max generation is reached or converged)
```

Figure 3. Complete GA

5 A* algorithm

A* [19] is a best-first search algorithm, which can guaranteed to find the optimal solutions. In a tree representation it starts from the root node, expands the intermediate nodes and finally reaches one of the leaf nodes. At each node, one of the aspects is assigned to a specific host as an addition to assignments made at its ancestors. Root node is a null solution of the problem. Intermediate nodes represent the partial solutions and leaf nodes represent the complete solutions.

Each node p in the tree maintains a cost function $f(p)$ which is computed as $f(p) = g(p) + h(p)$, where $g(p)$ is the cost of getting from the root to node p and $h(p)$ is the estimated cost of getting from p to the goal node. In our algorithm $g(p)$ is calculated using equations (1) and (2) as the load on the heaviest-loaded host (*F1*) of partial assignment. Since, at intermediate nodes all aspects have not been assigned yet, $g(p)$ is not sufficient solely to express the greatest load *F1*. Future assignments to the same host may increase this load. To be able to compare cost values of nodes in different levels fairly, possible effect of unassigned aspects on the load is added as $h(p)$ to $g(p)$. In our algorithm $h(p)$ is calculated as the sum of the object relation counts of aspects that are unassigned at node p. Let U is the set of unassigned aspects in node p, then $h(p)$ is calculated as follows:

$$h(p) = \sum_{i \in U} \left(\sum_{j=1}^{m} R_{ij} \right) \qquad (4)$$

Here $h(p)$ is not a real load value; it is just an estimation of the effect of future assignments that is used to compare cost values of different nodes fairly. Different functions may also be used as $h(p)$. In our study we chose the simple one in (4), which provides proper solutions.

As an illustration, for the sample system of three hosts, five objects and four aspects (see Figure 1) the resulting search tree of the A* algorithm is shown in Figure 4. A search-tree node includes partial assignment of aspects to hosts, and the value of the cost function. A partial assignment means that some aspects are unassigned; if there is an 'X' in the place of aspect a_i it indicates that i^{th} aspect has not been assigned yet. For example in Figure 4 the node with label 5 shows that aspect a_1 has been assigned to host 2, and the value produced by the cost function $f(p)$ is 485. The search tree's depth equals the number of aspects, and any node of the tree can have a maximum of n successors, which is the number of the hosts.

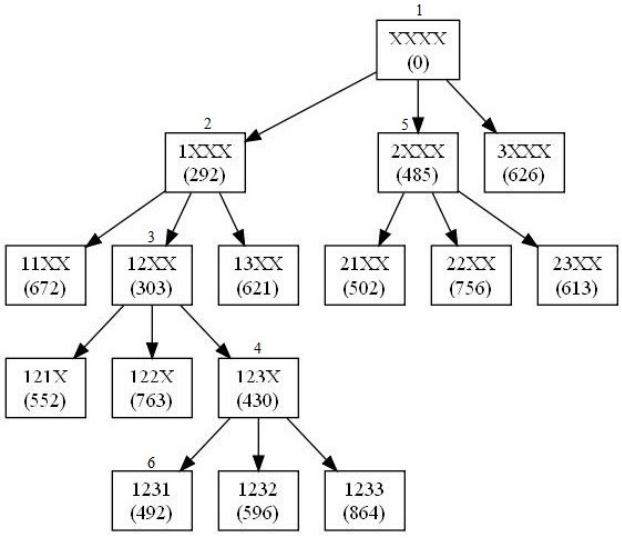


Figure 4. Search tree for A* algorithm

The algorithm maintains two lists named OPEN and CLOSED. The OPEN list keeps nodes that need to be examined, while the CLOSED list keeps nodes that have already been examined. When a node is selected from OPEN list to be examined, its child nodes are generated and put into the OPEN list. The nodes in the OPEN list are ordered before the selection according to cost function $f(p)$; that is, the algorithm selects the node with the minimum cost. Initially, the OPEN list contains just the root node, and the CLOSED list is empty.

In the example, given in Figure 4, labels show the selection order of the nodes for the given system. We start with the root node labeled as 1. We examine children of the root and select node 2 because it has the lowest cost value (292). Then all children of node 2 are added to the OPEN list, where children of root still exist. Now node 3 is selected from

the OPEN list because it has the smallest cost value and its children are added to the list. After that, nodes 4, 5 and finally 6 are selected from the OPEN list according to their cost values. Since node 6 is a leaf node the algorithm terminates and the final solution is the assignment of aspects as presented on this node. If more than one node have the same smallest cost value then the second objective function (*F2*) is taken into account, and the node with the smallest sum of load is selected. The complete A* algorithm is as follows:

```
Initialize OPEN and CLOSED lists
(OPEN=root node; CLOSED=EMPTY)
while the OPEN list is not empty {
  Get node p off the OPEN list with the lowest f(p)
  Add p to the CLOSED list
  if p is the leaf node then return p as solution
  Generate each successor node p' of p
  Add p' to the OPEN list
}
```

Figure 5. Complete A* algorithm

6 Experimental results

To evaluate the performance of our algorithms firstly, we coded our algorithms in Java programming language using Eclipse SDK 3.4.2 and compared their speeds for different aspect oriented programs by executing them on a server with four quad-core 2.60 GHz Intel Xeon CPU processors and 15 GB main memory, running the Ubuntu Linux 10.04.1. Secondly, aspects of different programs are assigned to hosts according to solutions obtained by two algorithms and these programs are executed on a simulation tool called the Asynchronous Distributed System Simulator [20]. We compared completion time of programs related to the different aspect assignments.

The Asynchronous Distributed System Simulator is written in Java programming language using a threaded architecture and can simulate any algorithm that has been designed for the distributed system network. It takes input parameters through an XML file which specifies the nodes in the network, the links between the hosts and the algorithm to be run on the distributed system. The simulator has a queue of messages that represents messages that are in transit on the network. Each link has a delay associated with it and messages sent using a link are not delivered until after the delay period has passed.

In our experiments we test our algorithms on a fully connected distributed system with five hosts. The host connectivity graph of the system is shown in Figure 6, where labels on edges show the cost of delays of the communication links (D_{pq}). On this system we try to distribute aspects of three aspect oriented programs with different sizes. The programs are detailed below:

P1 : 10 objects and 5 aspects
P2 : 20 objects and 10 aspects
P3 : 30 objects and 15 aspects

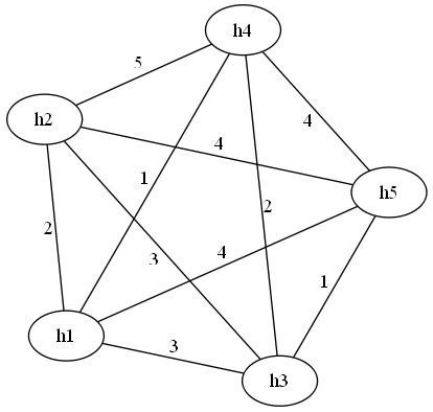


Figure 6. Host connectivity graph used in experiments

For each of these programs we generate 10 different datasets randomly, which include following properties of programs: aspect execution costs (X_{iq}), aspect-object relation counts (R_{ij}), aspect-object communication costs (C_{ij}) and object locations (L_j). X_{iq} and C_{ij} cost values are generated randomly in the range of [1, 10]. Similarly, R_{ij} values are generated randomly in the range of [0, 20]. These datasets can be found in [22].

Table 1. Obtained cost values execution times of two algorithms for P1 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	1690	6287	441	1690	6287	25
2	1627	6505	379	1627	6505	43
3	1713	6173	371	1713	6173	42
4	1615	7240	425	1615	7240	31
5	2012	5751	418	2012	5751	43
6	1838	7057	378	1838	7057	42
7	1427	5194	368	1427	5194	24
8	1763	7023	359	1763	7023	38
9	1985	6994	399	1985	6994	47
10	1354	5307	363	1354	5307	34
Average			390			37

Table 2. Obtained cost values execution times of two algorithms for P2 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	6836	31293	1392	6246	29085	671278
2	7013	30520	1289	6759	29237	441766
3	6895	29182	1342	6131	28809	317609
4	7110	32584	1373	7110	30025	524357
5	6604	27703	1349	6456	27601	389860
6	5388	22709	1211	5388	21194	478004
7	6412	28918	1301	5919	26361	383846
8	6127	28192	1307	6127	28051	680675
9	6716	29391	1153	6019	27461	238493
10	6758	27687	1291	6267	25851	174162
Average			1301			430005

Table 3. Obtained cost values execution times of two algorithms for P3 in milliseconds

# of dataset	GA			A*		
	F1	F2	Time	F1	F2	Time
1	14701	64427	3556	12078	56246	3935320
2	13672	61427	3642	12435	59121	9420174
3	14126	65993	3520	12871	63270	1956809
4	15334	69969	3387	13115	59602	5884341
5	13117	60436	3653	12045	55062	5229528
6	13725	65806	3549	12346	59445	3257284
7	13543	60250	3675	12053	56469	6977759
8	13667	64415	3487	12741	58318	7851088
9	16001	69825	3516	13734	65334	3139030
10	14588	66789	3520	13291	61567	2761528
Average			3551			5041286

Using the results of simulations we evaluate performance of our algorithms in two ways. Firstly, we consider execution times of algorithms, spent to find a solution. Secondly we investigate the completion time of the AOP, when the aspects are assigned to host according the solutions provided by the algorithms.

Tables 1, 2 and 3 provide performance comparison of two algorithms by considering obtained cost values (*F1* and *F2*) and their execution times for three different programs. Results in Table 1 show that GA and A* obtains always the same cost values for *P1*, which is a relative smaller program than *P2* and *P3*. In this case A* performs almost 10 times faster than GA. When the number of aspects increases A* obtains better (smaller) cost values than the GA as it is shown in Tables 2 and 3. This means that A* can distribute aspects more efficiently than the GA for bigger programs. A* achieves about 7% smaller *F1* values for *P2* and about 10% smaller values for *P3* compared to the GA. On the other hand, with the increase in the number of aspects and objects in the program, the execution time of A* increases very fast. For *P2* the GA proposes a solution 300 times faster and for *P3* nearly 1400 times faster than the A*.

The relation between the performance of the algorithms and the number of objects and aspects in the program can be explained as follows. A* algorithm uses a best-first search technique that builds a search-tree by visiting the most promising nodes first. When the number of nodes in the search tree is smaller it quickly reaches the solution node. But if the number of aspects increases, nodes in the tree also increase and the algorithm spends more time to visit these nodes. On the other hand GA uses a random search technique, which requires only a certain number of iterations to obtain a solution. Therefore if the number of aspects increase the execution time of the A* is increased much more that the GA. However it is expected that the A* can find optimal solution in all cases, while the GA can obtain optimal aspect assignments only for relative small systems.

In order to validate the efficiency of the aspect assignments of two algorithms we run three aspect oriented programs on the simulator and measure the completion time of these programs under different assignments of aspects. To evaluate the performance improvement achieved by our algorithms, we created a rival algorithm, namely the random assignment algorithm (RAA). The RAA assigns aspects to hosts randomly without taking any properties of the system and program into consideration. This is our baseline algorithm that helps us to observe the speedup obtained by the proposed algorithms. We performed the RAA on three programs (*P1*, *P2*, *P3*) for each dataset 10 times. We ran these programs on the simulator for 10 different random assignments produced by the RAA and calculated the average of the completion time T(RAA) for each dataset. To get the speedup of the AOPs we do the following calculations: T(RAA)/T(GA), and T(RAA)/T(A*), where T(GA) and T(A*) are completion times of the AOPs, when aspects are assigned according to the GA and A*, respectively. Results are given in Table 4. For example, the value 2.6 in the first row and column of the table denotes that the execution time of the AOP *P1*for dataset #1 takes 2.6 times longer if the aspects are assigned by the RAA then the case where aspect assignment is performed by the GA or A*.

Table 4. Speedup of programs using proposed algorithms relative to random assignment

# of dataset	P1	P2		P3	
	GA and A*	GA	A*	GA	A*
1	2.6	1.8	1.9	1.9	2.0
2	2.3	1.7	1.7	1.8	2.2
3	2.2	2.3	2.6	1.9	2.1
4	2.5	2.0	2.0	1.8	2.0
5	1.8	2.0	2.1	2.1	2.3
6	1.9	2.1	2.1	1.8	1.9
7	2.8	1.9	2.1	1.9	2.2
8	2.4	2.2	2.2	1.8	1.9
9	2.4	1.8	2.0	2.0	2.3
10	2.9	2.0	2.2	1.8	2.0

We deduce from Table 4 two main results. Firstly, properly assignment of aspects improves the performance of a distributed AOP. Experimental result show that the proposed algorithms can speed up the AOPs between 1.7 and 2.9 times. Secondly, we see that A* achieves slightly higher speedups then the GA except for *P1*, where the GA obtains also the same values. This result was expected, since the cost values (*F1*) given in Tables 1, 2 and 3 are related to the completion time of the AOPs and they have almost the same characteristic as the speedup values in Table 4.

7 Conclusion

In this paper we first formulate the aspect assignment problem for distributed AOP. During this formulation we consider properties of heterogeneous distributed systems and distributed AOPs, such as processing capabilities of hosts, delays of communication links, amount of transferred data between objects and related aspects. Then we propose two different algorithms to solve this problem. One of these

algorithms is GA which is based on the laws of natural evolution and the second one is A* algorithm which is based on best-first search.

Experimental results show that the proposed algorithms have their own advantages and disadvantages compared to each other. Firstly, we noticed that the A* algorithm obtained the optimal assignments for each of the programs with all datasets we used. On the other hand, the GA found the optimal assignments for small sized programs and sub-optimal solutions if the size of the programs increased. Secondly, the solution time for A* algorithm is considerably shorter than GA when the search space is smaller. However, the duration of the A* algorithm increases with the growth of the search space very fast and GA performs better, namely up to 1400 faster for one of the tested programs.

To evaluate proposed algorithms and examine the effect of assignment of aspects on the speed of the AOPs, we distributed aspects in three different ways, namely according to GA, A* and randomly. Then we compared the completion time of the AOPs under different aspect assignments The simulation results indicate that properly assignment of aspects can speed up the AOPs between 1.7 and 2.9 times. We also see that A* provides approximately 10% higher speedups then the GA for relatively larger programs. In conclusion, properly assignment of aspects improves performance of the distributed AOPs, and because it's shorter response times the proposed GA can be preferred to solve this assignment problem.

8 References

[1] T. Elrad, R. E. Filman, and A. Bader, "Aspect-oriented programming," *Communications of the ACM*, Vol. 44, No. 10, October 2001, pp. 29-32.

[2] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold, "An overview of AspectJ," *In 15th European Conference on Object-Oriented Programming (ECOOP 2001)*, Budapest, Hungary, June 2001, pp. 327–353

[3] AspectWerkz, "AspectWerkz Dynamic AOP for Java Overview", http://aspectwerkz.codehaus.org/, 2004.

[4] B. Burke and al. JBoss-AOP. www.jboss.org/developers/projects/jboss/aop.

[5] PostSharp. http://www.postsharp.org/.

[6] R. Johnson et al. *Spring – Java / J2EE application framework. Reference Manual Version 2.0.6*, Interface21 Ltd., 2007.

[7] M. Nishizawa, S. Chiba, and M. Tatsubori. "Remote pointcut – a language construct for distributed AOP," *Proc. ACM Int'l Aspect Oriented Software Development*, 2004, pp. 7-15.

[8] R. Pawlak, L. Seinturier, L. Duchien, G. Florin, F. Legond-Aubry, and L. Martelli, "JAC: an aspect-oriented distributed dynamic framework," *Software: Practice and Experience*, Vol. 34, No. 12, 2004, pp. 1119–1148.

[9] L. D. Benavides Navarro, M. Südholt, W. Vanderperren, B. De Fraine, and D. Suvèe. "Explicitly distributed AOP using AWED," *In Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD 2006)*, Bonn, Germany, March 2006, pp. 51–62.

[10] R. Mond´ejar, P. Garc´ıa, C. Pairot, and A. Skarmeta. "Building a distributed AOP middleware for large scale systems," *In Proceedings of the 2008 Workshop on Next Generation Aspect Oriented Middleware (NAOMI 08)*, New York, USA, April 2008, pp. 17–22.

[11] É. Tanter and R. Toledo. "A Versatile Kernel for Distributed AOP," *In Proceedings of the IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 2006)*, Bologna, Italy, June 2006, pp. 316–331.

[12] M. Gursky, "Some complexity results for a multi-processor scheduling problem," Private Communication from H. S. Stone, 1981.

[13] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Eng.*, Vol. SE-3, 1977, pp. 85-93.

[14] W. W. Chu, "Optimal file allocation in multiple computing system," *IEEE Trans. Comp.*, Vol. C-18, 1969, pp. 885-889.

[15] V. M. Lo, "Heuristic algorithms for task assignment in distributed systems," *IEEE Trans. On Computers*, Vol. 37, No. 11, 1988.

[16] B. Shirazi, M. Wang and G. Pathak, "Analysis and evaluation of heuristic methods for static task scheduling," *J. Parallel Distrib. Comp.*, Vol. 10, 1990, pp. 222-232.

[17] S.S. Wu and D. Sweeping, "Heuristic Algorithms for Task Assignment and Scheduling in a Processor Network," *Parallel Computing*, Vol. 20, 1994, pp. 1-14.

[18] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Massachusetts: Addison Wesley, 1989.

[19] P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, Vol. 4, No. 2, 1968, pp. 100-107.

[20] S. Burgess, *Asynchronous Distributed System Simulator*, University of Western Ontario, Computer Science, 2007.

[21] Obtaining crosscutting relationship information from AJDT, http://wiki.eclipse.org/Developer's_guide_to_ building_tools_on_top_of_AJDT_and_AspectJ

[22] http://web.itu.edu.tr/bulusa/DS/

Using ATL Transformations to Derive RSL Specifications from Feature Models

L. Felice, M. Ridao, M.V. Mauco and M.C. Leonardi

INTIA. Facultad de Ciencias Exactas
Universidad Nacional del Centro de la Provincia de Buenos Aires, Tandil, Argentina

Abstract - *Model Driven Development (MDD) is a software development paradigm based on automatic transformations of models. We have proposed a strategy to enhance the RAISE formal method by integrating the phase reusable Domain Analysis using Feature Models in the early steps of software development process. The strategy defines a set of mappings from Feature Model elements to RAISE Specification Language (RSL) constructs. In order to fit this strategy into MDD, we present in this paper an ATL transformation which allows the automatic derivation of a first RSL specification. The ATL rules formalize the mappings, and define how features and relationships between them (source model) are matched and navigated to produce the RSL specification (target model).*

Keywords: Model Driven Development, Feature Model, RAISE Method, ATL Transformation Language.

1 Introduction

Formal methods have come into use for the construction of real systems, as they help to increase software quality and reliability. When used early in the software development process, they can reveal ambiguities, incompleteness, inconsistencies, errors or misunderstandings that otherwise might be only discovered during costly testing and debugging phases. The RAISE Method [5], for example, is intended for use on real developments, and includes a large number of techniques and strategies for doing formal development and proofs, as well as a formal specification language, the RAISE Specification Language (RSL) [4], and a set of tools to help writing, checking, printing, storing, transforming, and reasoning about specifications. However, formal specifications are unfamiliar to stakeholders, whose active participation is crucial in the first stages of software development process to understand and communicate the problem. This holds in Domain Analysis (DA), because its first stage is to capture the knowledge of a particular domain, making necessary to have a model that is comprehensible by software engineers and domain experts.

To contribute to bridge this gap, we have been working in the integration of a domain analysis phase into the RAISE Method, in order to specify a family of systems to produce qualitative and reliable applications in a domain, promoting early reuse and reducing development costs. We proposed to use feature models to represent the domain analysis because they facilitate the customization of software requirements. In DA, features and relationships between features (called domain feature model) are used to organize the requirements of a set of similar applications in a software domain. We have first proposed an informal strategy which starts by defining this feature model following one of the several proposals that facilitate feature models' construction: Feature-Oriented Reuse Method (FORM) [10]. This model is then transformed, by means of a set of manual heuristics, into a RSL specification that can be later developed into a more concrete one to automatically obtain a prototype to validate the specification by using the RAISE Tools [6]. The use of a feature model is motivated by the fact that stakeholders often speak about product characteristics in terms of "features the product has and/or delivers", using them to communicate their ideas, needs, and problems.

The Model Driven Development [11], known as MDD, is a new software development paradigm. MDD stresses the use of models in the software development life cycle and argues automation via model execution, model transformation and code generation techniques. One of the key features of this paradigm is the notion of automatic transformations that describe how a model in a source language (source model) can be transformed into one or more models in a target language (target model). In order to fit our proposal of enhancement of formal developments with the RAISE Method into MDD paradigm, we present in this paper an ATL [1] transformation which allows the automatic derivation of a first abstract RSL specification of a domain starting from a feature model. The ATL rules formalize the heuristics presented in [3], and define how features and relationships between them (source model) are matched and navigated to produce the RSL specification (target model). The rules follow closely the principles proposed in the RAISE Method, so this first and still incomplete specification may be later developed into a concrete one following the RAISE Method steps. With a concrete specification, the RAISE tools can be used to

automatically obtain a quick prototype and get a feeling of what the specification really does.

The paper is organized as follows. Section 2 gives an overview of FORM, the source model; while the target model, RSL specifications, is presented in Section 3. The core of the paper is in Section 4, where we describe the transformation strategy proposed and some of the ATL rules defined. Finally, in Section 5 we present some conclusions and outline possible future works.

2 The Source Model: Feature Model (FORM)

A domain model is the result of the analysis of commonalities and variabilities of systems within a domain. It is a high level description of the application family, providing a framework for describing the essential characteristics. Examples of more relevant DA methods include Feature-Oriented Domain Analysis (FODA) [7], Organization Domain Modeling (ODM) [13], FeatureRSEB [7] and FORM [10]. They support the notion of feature-oriented. This is a concept based on the emphasis this method places on finding the features or functionalities usually expected in applications for a given domain. Feature Models are a modeling notation to represent the variability in a system family and describe all valid configurations.

The model constructed captures commonality as an AND/OR graph. Then, this model is used to define parameterized reference architectures and appropriate reusable components which are instantiated during actual application development. Feature models are able to describe the aspects, commonalities or characteristics of a system and include variability modeling for system families. Commonalities can be modeled by common features (mandatory features whose ancestors are also mandatory), and variabilities can be modeled by variant features, such as optional, alternative, and or-features.

A feature is a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems [9]. A feature is detailed in any number of other features (subfeatures). Mandatory features describe detailed aspects that the parent feature must support, while optional features may be selected when creating a concrete system from a feature model. Alternative features have a multiplicity similar to the UML multiplicity, defining how many of the features must or may be selected. In addition to the hierarchical relationship, a constraint relation defines if a feature requires, modifies or excludes with any other features. Figure 1 summarizes the notation used in [12].

The feature requested by a stakeholder is called a concept feature, it is a root node of the feature diagram and all features are represented as child nodes. The hierarchical relationships mandatory, optional, and alternative are represented by different edges between the nodes. A simple line with a filled circle represents a mandatory relationship, while the optional is represented with a line ending with an empty circle. Arcs spanning two or more edges of the feature nodes depict a set of alternative features. The arc is annotated with the multiplicity of the alternative. Normally, a grouped feature has [0..1] as its default cardinality. The cardinalities [1..1] (xor group), [1..k] (or group) and [0..0] are used for features that were selected or eliminated, respectively, from a group during specialization.

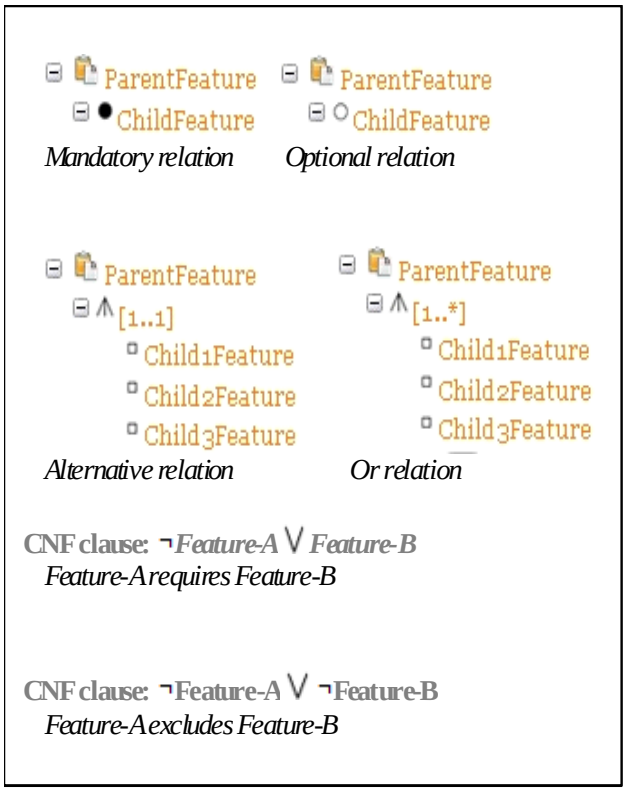


Figure 1. Feature Model Notations

Based on the concepts described above, a metamodel of the FORM feature-based method is presented in Figure 2. The metamodel is expressed using feature modeling based on associations between the features in a concise way. This metamodel is the source model to the ATL transformations of a feature-based model to RSL specifications.

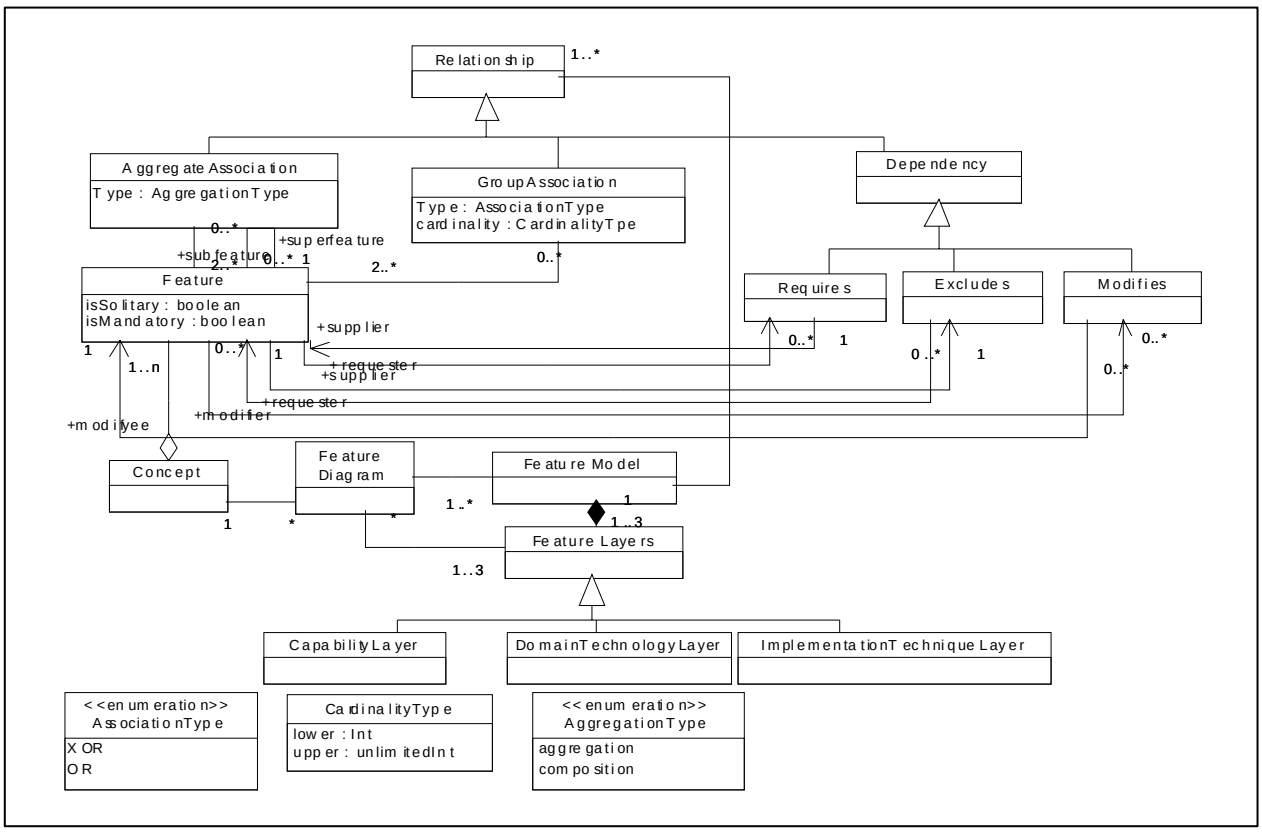


Figure 2. FORM metamodel

3 The Target Model: RSL Specifications

RAISE gives its name to a wide spectrum specification and design language, the RAISE Specification Language (RSL), an associated method, and an available set of tools to help writing, checking, printing, storing, transforming, and reasoning about specifications. Complete descriptions of RSL and the RAISE Method can be found in the corresponding books [4] and [5], while the tools are described in [6] and they can be downloaded from UNU-IIST's web site (www.iist.unu.edu).

There are two main activities in the RAISE method: writing an initial specification, and developing it towards something that can be implemented in a programming language. Usually the first RSL specification is an abstract, applicative and sequential one, which is later developed into a concrete specification, initially still applicative and then, imperative, and sometimes concurrent. The method provides many guidelines to hierarchically structure a specification, aiming at encouraging separate development and step-wise development. A specification in RSL is a collection of modules. A module is basically a named collection of declarations; it can be a scheme or an object. Each module should have only one type of interest, defining the appropriate functions to create, modify, and observe values of the type. However, the kernel module concept is that of a class expression. A basic class expression is a collection of declarations enclosed by the keywords class and end and it represents a class of models. Objects and schemes are defined using class expressions. A typical applicative class expression contains type, value, and some axiom definitions. Axioms may be used to constrain the values or to model invariants.

RSL is a typed language. This means each occurrence of an identifier representing a value, variable, or channel must be associated with a unique type. Besides, it must be possible to check whether each occurrence of an identifier is consistent with a collection of typing rules. A type is a collection of logically related values, and it may be specified by an abstract or a concrete definition. An abstract type, also referred to as a sort, has only a name. It is a type we need but whose definition we have not decided yet. A concrete type can be defined as being equal to some other type or using a type expression formed from other types. Values are constants and functions. Their definition must include at least the signature that is a name, and types for the result, and for the arguments, in case of a function.

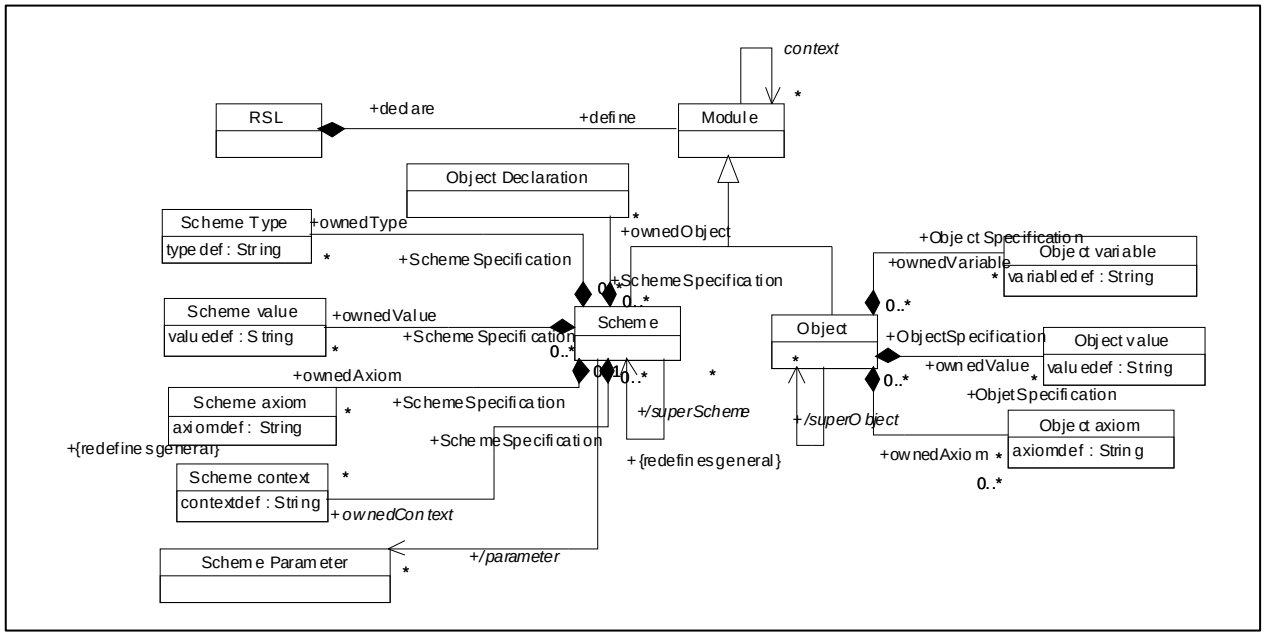


Figure 3. RSL metamodel

Figure 3 depicts the RSL metamodel defined for the ATL transformation. The elements included in the metamodel are the ones involved in the mappings defined by the strategy.

4 The Strategy to Derive RSL Specifications from Feature Models

In this section we discuss how Feature Modeling serves as a guideline to identify RSL reusable specifications. We use the structural information of the Feature Models to derive RSL constructs based on [2] where the features are typed. Mandatory features, Optional features, OR features and alternative features must be differentiated. We propose a mapping to derive an initial hierarchy of RSL types from a feature model. The goal of each rule is to identify an element or a combination of connected elements of the feature model and give the equivalences in terms of RSL constructs.

To follow one of the principles proposed in the RAISE Method, schemes are identified by the rules obtaining a set of RSL abstract types, that later are developed into more concrete ones. The rules are defined to describe schemes analyzing the feature model in order to complete the specification of each type, defining attributes, cardinalities, and axioms. The main objective is to map a feature model into a RSL scheme hierarchy. To do this, it is neccesary to transform each diagram of the model. We describe how to map each feature of the diagrams, its relationships and dependencies among them. Thus, the transformation rules are defined by the following mappings:

- Root feature (Concept) is mapped into a RSL scheme with a type of interest (Conc2Scheme).
- Feature is mapped into a RSL module.
  - 'solitary', 'mandatory' and 'optional' features into a RSL scheme (Fea2Scheme)
  - 'Capability layer' feature into a RSL scheme with functions declaration (FOP2Scheme)
- Relationship
  - *Dependencies* between features are mapped into RSL axioms expressing the corresponding restriction (Dp2Axiom)
    - Requires dependency (Requires2Axiom)
    - Excludes dependency (Excludes2Axiom)
  - *Aggregate* is mapped into RSL objects distinguishing between whole objects and part objects (Agg2Scheme)
  - *Grouping* is mapped into RSL schemes relations (Group2Scheme)
    - Grouping OR (GroupOR2Scheme)
    - Grouping XOR (GroupXOR2Scheme)
- Diagram feature is mapped into a RSL module hierarchy (FD2Hierarchy)
- Feature Model is mapped into one or more RSL module hierarchies (FM2Hierarchies)

4.1 Model Transformation Rules

The most usual form for the definition of mappings is the use of natural language. This tends to be a rather vague form of definition, often relying on some examples to explain the meaning. While this might be intuitive to humans, it completely lacks the formal definition needed for automatic processing. Completeness and unambiguity are hard to reach with natural language. This kind of mapping has been recognized to be inadequate to manage the multitude of models that are produced in a software development process [8]. For these reasons, we model the transformation rules as packages having a source and a target pattern inside. Correspondence between source and target elements are denoted by pentagons.

The complete strategy consisting of 20 rules and a case study is found in [3]. In this paper we describe a subset of the mappings proposed and the corresponding transformation rule written in ATL [1]: the tranformations for the *Conc2Scheme*, *Fea2Scheme* and *Dp2Axiom* mappings.

Conc2Scheme

Description: The concept is the root feature of the feature diagram. It represents the system domain we are modeling and it is the element with highest hierarchy in a diagram. Each concept c in a Feature Model is mapped into a RSL scheme with the same name. This scheme will have a type of interest. The transformation rule is shown in figure 4.

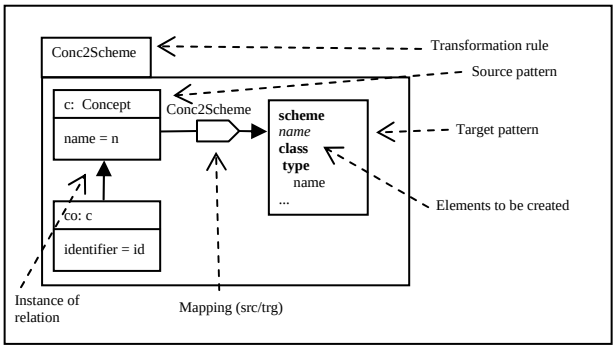


Figure 4. Conc2Scheme transformation

Fea2Scheme

Description: A feature which verifies the isSolitary property, expresses that it is a specialization of its super feature, being a leaf in the diagram. Also, features could be mandatory or optional (isMandatory property) preserving its super feature property. The RSL specification will express these properties in the value and extend clauses. So, it is neccesary to create a super scheme. If the super scheme is a concept it will not be neccesary to define it because the Conc2Scheme mapping has defined it before. Figure 5 represents the transformation rule for this mapping.

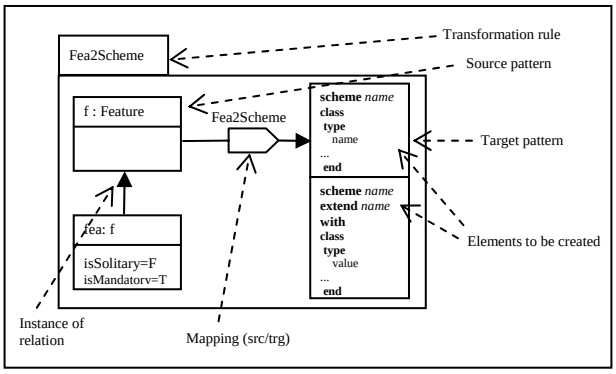


Figure 5. Fea2Scheme transformation

Dp2Axiom

Requires2Axiom Description: when a feature A requires a feature B, the relation 'requires' reflects that if A is selected then B must be selected too. The RSL specification will contain an axiom to specify this relation by means of the function is_selected.

Excludes2Axiom Description: when a feature A excludes a feature B, the relation 'excludes' reflects that if A is selected, then B must not be selected. The RSL specification will contain an axiom to specify this relation by means of the function ~is_selected.

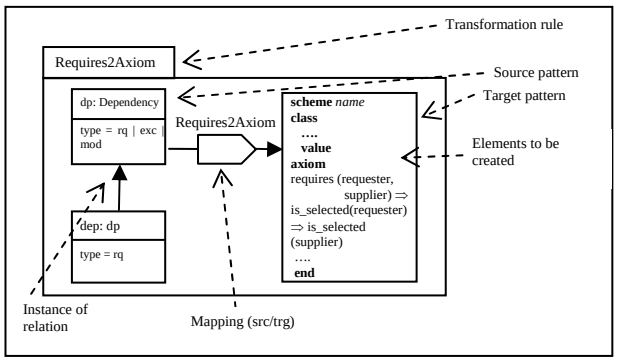


Figure 6. Requires2Axiom transformation

Figure 6 represents the first mapping. For Excludes2Axiom mapping the axiom excludes expresses that B must no selected if A is selected.

4.2 Model Transformation Rules in ATL

In this section we present the rules, written in ATL, to transform a feature model (source model) into RSL specifications (target model). The definition in ATL is a module that involves several ATL rules (like lazy and matched rules) with a set of helpers.

We only present the rules for the subset of mappings described above (Conc2Scheme, Fea2Scheme,

Requires2Axiom and Excludes2Axiom). The transformation module FM2RSL names the variables corresponding to the source and target models ("IN" and "OUT") together with their metamodels ("FM" and "RSL") acting as types. Next come helpers and lazy rules necessary to compute a result to be used in the rules Conc2Scheme, LeafFea2Scheme, and Fea2Scheme.

```
module FM2RSL;
create OUT: RSL from IN: FM;

helper context FM!Feature def : getValueIMDef :
String =
if self.IsMandatory
then
   F.name + 'isMandatory = TRUE'
else
   F.name + 'isMandatory = FALSE'
endif;

helper context FM!Feature def : getValueISDef :
String =
if self.IsSolitary
then
   F.name + 'isSolitary = TRUE'
else
   F.name + 'isSolitary = FALSE'
endif;

helper context FM!Feature def : requires :
Set(FM!Requires)=
if not self.requester.oclIsUndefined()
then
   self.requester -> collect( requires |
requires )
else
   Set{}
endif;

helper context FM!Feature def : excludes :
Set(FM!Excludes)=
if not self.requester.oclIsUndefined()
then
   self.requester -> collect( excludes |
excludes )
else
   Set{}
endif;

...

unique lazy rule Concept2Type {
from
   C: FM!Concept
to
   ST: RSL!SchemeType (
   typedef <- C.name )
}

unique lazy rule Feature2ValueIsMandatory {
from
   F: FM!Feature
to
   SV: RSL!SchemeValue (
   valuedef <- F.getValueIMDef )
}

unique lazy rule Feature2ValueIsSolitary {
from
   F: FM!Feature
to
   SV: RSL!SchemeValue (
   valuedef <- F.getValueISDef )
}

unique lazy rule Feature2TypeIsSolitary {
from
   F: FM!Feature
to
   SA: RSL!SchemeType (
   typedef <- F.name + ':: isSolitary: ' +
   F.name + 'isSolitary' )
}

unique lazy rule Feature2TypeIsMandatory {
from
   F: FM!Feature
to
   SA: RSL!SchemeType (
   typedef <- F.name + ':: isMandatory: ' +
   F.name + 'isMandatory' )
}

unique lazy rule Feature2ContextExtend {
from
   F: FM!Feature
to
   SC: RSL!SchemeContext (
   contextdef <- 'extend ' + F.superfeature )
}

unique lazy rule Requires2Context {
from
   R: FM!Requires
to
   SC: RSL!SchemeContext (
   contextdef <- R.supplier.name )
}

unique lazy rule Requires2Axiom {
from
   R: FM!Requires
to
   SA: RSL!SchemeAxiom (
   axiomdef <- '(' + R.requester.name + ',' +
   R.supplier.name + ') ≡ is_selected(' +
   R.requester.name + ') => is_selected(' +
   R.supplier.name + ')' )
}

unique lazy rule Excludes2Context {
from
   E: FM!Excludes
to
   SC: RSL!SchemeContext (
   contextdef <- E.supplier.name )
}

unique lazy rule Excludes2Axiom {
from
   E: FM!Excludes
to
   SA: RSL!SchemeAxiom (
   axiomdef <- '(' + R.requester.name + ',' +
   R.supplier.name + ') ≡ is_selected(' +
   R.requester.name + ') => ~is_selected(' +
   R.supplier.name + ')')
}

...

rule Conc2Scheme {
from
   C: FM!Concept
to
   S: RSL!Scheme (
   name <- toUpper(C.name),
   ownedType <- ThisModule.Concept2Type(C) )
}
```

```
rule LeafFea2Scheme {
from
   F: FM!Feature (F.IsSolitary = true)
to
   S: RSL!Scheme (
   name <- toUpper(F.name),
   ownedType <-
   thisModule.Feature2TypeIsSolitary(F),
   ownedType <-
   thisModule.Feature2TypeIsMandatory(F),
   ownedValue <-
   thisModule.Feature2ValueIsSolitary(F),
   ownedValue <-
   thisModule.Feature2ValueIsMandatory(F),
   ownedContext <-
   thisModule.Feature2ContextExtend(F),
   ownedContext <- F.requires -> collect
   (requires |
   thisModule.Requires2Context(requires)),
   ownedAxiom <- F.requires -> collect
   (requires |
   thisModule.Requires2Axiom(requires))
   ownedContext <- F.excludes -> collect
   (excludes |
   thisModule.Excludes2Context(excludes)),
   ownedAxiom <- F.excludes -> collect
   (excludes |
   thisModule.Excludes2Axiom(excludes))
   SuperScheme <- F.superfeature.feature )
}
rule Fea2Scheme {
from
   F: FM!Feature (F.IsSolitary = false and not
   F.subfeature.oclIsUndefined())
to
   ST: RSL!Scheme (
   name <- F.name,
   ownedType <-
   thisModule.Feature2TypeIsSolitary(F),
   ownedType <-
   thisModule.Feature2TypeIsMandatory(F),
   ownedValue <-
   thisModule.Feature2ValueIsSolitary(F),
   ownedValue <-
   thisModule.Feature2ValueIsMandatory(F),
   ownedContext <- F.requires -> collect
   (requires |
   thisModule.Requires2Context(requires)),
   ownedAxiom <- F.requires -> collect
   (requires |
   thisModule.Requires2Axiom(requires))
   ownedContext <- F.excludes -> collect
   (excludes |
   thisModule.Excludes2Context(excludes)),
   ownedAxiom <- F.excludes -> collect
   (excludes |
   thisModule.Excludes2Axiom(excludes)) )
}
```

5 References

[1] ATL Transformation Language. http://www.eclipse.org/atl/. Last access April 2011.

[2] K. Czarnecki, S. Helsen, and U. Eisenecker. "Staged Configuration of Feature Models". Software Process Improvement and Practice, v.10, Nº 2, pp. 143-169, 2005.

[4] L.Felice, C. Leonardi, V. Mauco, G. Montejano, D. Riesco and N. Debnath. "Integrating Formal Methods with Domain Analysis". Journal of Computational Methods in Sciences and Engineering. Volume 10 Issue 1-2S2, pp. 149-161, September 2010.

[5] C. George, P. Haff, K. Havelund, A. Haxthausen, R. Milne, C.B. Nielsen, S. Prehn, S., and K.R.Wagner. The RAISE Specification Language. Prentice Hall, 1992.

[6] C. George, A. Haxthausen, S. Hughes, R. Milne, S. Prehn and J.S. Pedersen. The RAISE Development Method. BCS Practitioner Series, Prentice Hall, 1995.

[7] C. George. RAISE Tools User Guide, UNU/IIST, Macau, Research Report 227. 2001. http://www.iist.unu.edu. Last access April 2011.

[8] D. Griss, R. Allen and M. d´Alessandro. "Integrating Feature Modelling with the RSEB". Proceedings of the 5th International Conference of Software Reuse (ICSR-5). pp. 76-85, 1998.

[9] J. H. Hausmann and S. Kent. Visualizing model mappings in UML. In SoftVis 03: ACM Symp. on Software Visualization, pp. 169–178. San Diego, CA, USA, 2003.

[10] K. Kang, S. Cohen, J. Hess, W. Novak and A. Peterson. Feature-oriented domain analysis feasibility study, Technical Report, Carnegie Mellon University, Software Engineering Institute, CMU/SE-90-TR-21, 1990.

[11] K. Kang, S. Kim, J. Lee and K. Kim. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering, pp.143-268, 1998.

[12] Mellor, S., Clark, A., Futagami, T.: Model-driven development. IEEE software 20(5) ,2003.

[13] I. Montero, Joaquin Pena, Antonio Ruiz-Cortes, "From Feature Models to Business Processes," scc, vol. 2, pp.605-608, 2008 IEEE International Conference on Services Computing Vol. 2, 2008.

[14] M. Simos. "Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle". Proceedings of the 1995 Symposium on Software reusability. USA. pp. 196-205, 1995..

A Model-Driven Development Approach for Modeling Business Transactions at the Computation Independent Model Level

José Bocanegra[1] **and Diego Castaño**[1]
[1]IT Department, Universidad de la Amazonia, Florencia, Caquetá, Colombia

Abstract— *With the globalization of markets, business transactions are gaining significant importance because they allow an abstract view of the interactions among organizations that work to fulfill their business goals. This situation has allowed the development of several research works dealing with the modeling of business transaction. However, none of this work takes Computational Independent Models (CIM) that focus on observing the interaction between companies from the point of view of the business managers.*

M. Papazoglou proposes a model that brings together all the relevant information for this type of interaction called Business Transaction Model (BTM) whose characteristics make it optimal for the CIM level. However, the author does not propose a graphical notation of this model. Moreover, the model has not the rigor necessary for an implementation based on Model-Driven Development (MDD).

In this paper, we propose a meta-model adapted to MDD and a notation based on UML2 collaborations with small extensions in order to instantiate the BTM and we develop a MDD transformation that can map some features of BTM to business process using the Business Process Modelling Notation (BPMN) and vice versa.

Keywords: Business Transactions, MDD, CIM

1. Introduction

The current economic landscape means that companies have to be highly competitive and improve their production processes. This context requires companies to make extensive use of information technology to support their own business processes and streamline the complex interactions that occur with their trading partners. M. Papazoglou has referred to these interactions as business transactions [5] defined as follows: *a business transaction is defined as a trading interaction between possibly multiple parties that strives to accomplish an explicitly shared business objective, which extends over a possibly long period of time and which is terminated successfully only upon recognition of the agreed conclusions between the interacting parties*.

Business transactions, aditionally to modeling the interaction between two or more organizations, also depicts high-level information for business managers.

Specifically, in [5], M. Papazoglou proposes a model called Business Transaction Model (thereafter BTM). In BTM, the author, additionally to business process, includes five additional elements to provide more expressiveness to the CIM model: (i) the parties and roles, (ii) constraints and invariants, (iii) documents, (iv) pre-defined business functions and (v) a set of attributes and relationships. A more detailed explanation of this proposal can be seen in Section 2.

In order to clarify the differences between business process models and CIM models, we present an example of a business transaction. In Figure 1, we can observe the flow of information and goods originating from a travel agency and an airline when a traveler decides to purchase a travel plan. Figure 1 is drawn using BPMN (Business Process Modeling Notation).

As we can see in Figure 1, in this type of model it is possible to visualize the activities, the order in which they are performed, and the involved actors. In this example, we see that the role *Traveler* performs the activity *Find Travel* and then pass control to the role *Travel agency*. In a business process we can also see the documents exchanged. For example, the *Traveler* provides itinerary information to the *Travel agency*. However, the main elements for a business transaction defined by Papazoglou, such as legal restrictions, economic conditions or the profiles of the roles are not taken into account. In Table 1, we can see an example of a business transaction that complements the business process of this example. Thus, we can observe the constraints limiting the role *Airline*, in this case, that is low cost. Likewise we can observe the business operations that should be able to provide the role *Travel agency*: *search plan* and *deliver plan*.

Table 1: Textual representation of a business transaction based in BTM.

Element	Description
Business transaction	Provide a travel plan
Roles	Airline, Traveler, Travel agency
Constraints	Only low cost airlines
Documents	Travel itinerary, tickets
Business operations: role travel agency	Find travel plan send plan

Although the model BTM improve the state of the art of models used at CIM level, the author does not detail several important respects: (i) does not propose a graphical notation for the meta-model instantiation, (ii) does not show the correlation between business processes (taken into account in

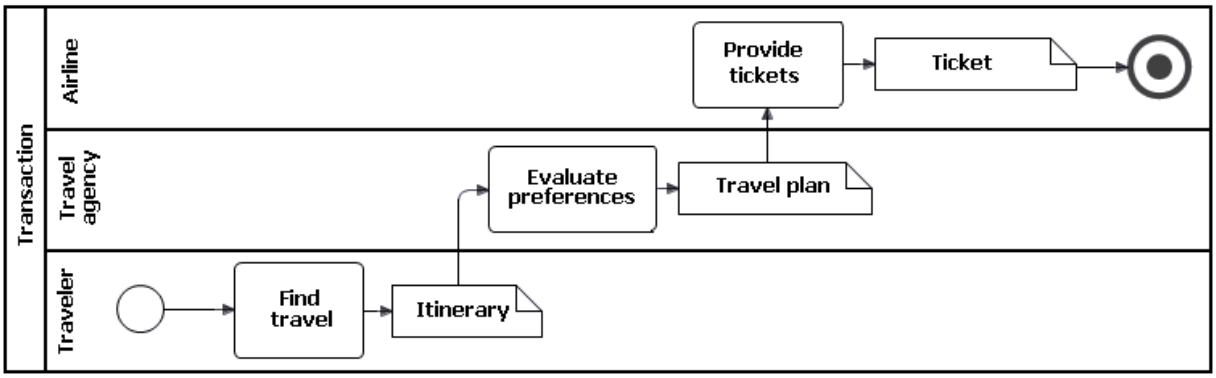


Fig. 1: Business process in BPMN.

the meta-model) and transaction models, (iii) the meta-model has some drawbacks as it was not tested in a deployment environment, and (iv) does not define MDD transformations from source model to other models such as CIM, PIM or PSM, but it suggests, without going into detail, some possible correlations with standards for web services. Respect to other proposals for modeling at CIM level, they have focused only on modeling business processes, leaving out important information that should be represented in a model of transactions, eg [3], [9], [1]. These problems, motivate us to complement the model proposed by M. Papazoglou and its correlation with the imple- mentation process using a MDD approach. For this reason, we present the following contributions:

- We provide a meta-model adapted to MDD and we provide a notation based on UML2 collaborations with small extensions in order to instantiate BTM (Section 3). In this section, we present some drawbacks in the meta- model of BTM, and we made some improvements.
- We develop a MDD transformation that allows to map some static features from BTM to business processes using BPMN notation and vice versa. This allows us keep both views consistent with each other (Section 4).
- To validate both the model and the transformation we have developed a prototype presented in Section 5, using the ATL transformation language, and *Eclipse Graphical Modeling Framework* (GMF).

2. Related work

In this section we analyze the proposals that have addressed the modeling of business transactions. The summary of this analysis is depicted in Table 2.

2.1 CIM for business transactions

In order to determinate that information should be available in the CIM model for a business transaction, we take the proposal made by M. Papazoglou in [5]. Figure 2, represents the UML meta-model of the proposal, where each packet refers to an item:

- The party or parties that interact in the transaction and the roles played by each one (Package 1). i.e. *Airline*, *Travel* and *Travel agency*.
- Constraints and invariants that can be summarized as rules and constraints that are accepted by mutual agreement between the parties (Package 2). i.e. the role played by the *Airline* must be exclusively of low cost.
- The documents used in the transaction (Package 3). i.e. travel itinerary provided by the traveler.
- A set of predefined business functions. A business function is a description of business principles clearly defined (Package 4). i.e. a predefined function to order a travel plan.
- The processes performed in the interaction (Pack 5).
- Business operations and a set of attributes and relationships (Package 6). Example: search plan, plan to deliver.

However, and despite being one of the most comprehensive approaches, the author does not proposes a graphical notation to instantiate the model. This situation creates difficulties for automated processing. Additionally, there are other drawbacks that must be corrected, including the fact that the meta-model is not set attributes of classes, for example, to determine the information associated with a role and the existence of overlapping relationships between two classes, example, among the class BusinessTransaction and class Party. For this reason it is necessary extend this model to support the application of MDD techniques.

Other proposals, as presented in [9] covers business process. Unfortunately, the proposed models are located at the PIM level, that is more related to software building that to management of organizations. As shown in Table 2, the other proposals under discussion are limited to only model the business process.

2.2 Transformations

The proposal of M. Papazoglou mention a set of WS-* standards that can support the implementation of the trans-

Feature	Item	Proposals				
		[5]	[3]	[9]	[1]	Our approach
CIM Model	Parts and roles	√	∼	-	-	√
	Constraints and invariants	√	-	-	-	√
	Objects	√	-	∼	-	√
	Business functions	√	-	∼	-	√
	Business processes	√	√	∼	√	√
	Attribs	√	-	-	-	√
	Graphical notation	-	-	-	-	√
Transformations	From static model to dynamic model	-	-	-	-	√

Table 2: Proposals comparative

action. Unfortunately there are no details on the mapping between the model and standards.

In [3], Lopez et Al. propose a division of the business transaction at several levels: CIM, PIM, and PSM. This proposal already mentioned the issue of MDD, and how this can help in the process of transformation. The proposal raises a number of changes and MDD: from PIM to PSM, from PIM to PIM, and from PSM to PSM. The data model maps to a XML schema. The navigation model is mapped to a standard XML/HTML. The use case models of the transaction models are mapped to WSDL/BPEL. Unfortunately, this proposal does not provide the model transformations between CIM and PIM or PSM transformations between models and text.

In [1], Ibrahim et Al. propose three levels of division of the transaction: organizational, business and technical. Although the proposal does not contain elements of MDD, we assume that the three levels of the transaction can be regarded as three levels proposed by MDD: CIM (organizational level), PIM (business level) and PSM (technical level).

This allows us to assert that the proposed transformation is an improvement on the state of art.

3. Notation for BTM

We have taken as a reference for modeling business transactions the proposal of M. Papazoglou. Although the author does not suggest a graphical notation for instantiating the model, we chose to use UML2 models of collaboration. This decision is motivated because in the field of software agents that organize actors imitating people's organizations, one of elements to model agents and their interactions is UML collaborations [6]. In contrast, other notations as those based on use cases does not allow you to specify the necessary information without making large extensions.

3.1 Extensions and modifications to BTM and UML2 collaborations

Note that the inclusion of the notation has motivated us to make some modifications to the original meta-model [6]. These changes have required:

- We include a class named *Organization*. This allows grouping of roles/actors and organizations.
- Include two types of *Business Objects* (In and Out). This allows obtain the order of execution of activities. As noted, we complement BTM adding that product infor- mation/documents/services are necessary for a transaction and goods/documents/services are produced.

Regarding the UML collaborations, we have developed a set of modifications based on earlier works by one of the authors [6], [7], [8]. The proposed extensions are:

- We split the collaboration icons into three compartments to include the following information: (i) the type of collaboration (transaction or activity), (ii) the name of the transaction or activity, and (iii) the documents handled.
- We split the *CollaborationsRoles* into three compartments to include the following information: (i) the name of the role, (ii) the documents handled, and (iii) the business functions or skills offered by the role in the transaction.

3.2 Notation for static models of BTM

The following is the notation that we used for the instantiation model:

- A business transaction is represented by a UML2 collaboration that presents a graphical notation as a dotted ellipse divided into three compartments. In the first com- partment located the name of the transaction, the second a description, and the third, the documents associated with that transaction.
- The roles are represented using CollaborationsRoles of UML2. In CollaborationsRoles we include the following: (i) the name of the role, (ii) documents/products handled and (iii) the transactions executed.
- Business constraints are represented by an expression in square brackets and placed in a note attached to collaboration icon.
- Business objects are represented by compartments in both roles as the business transaction.

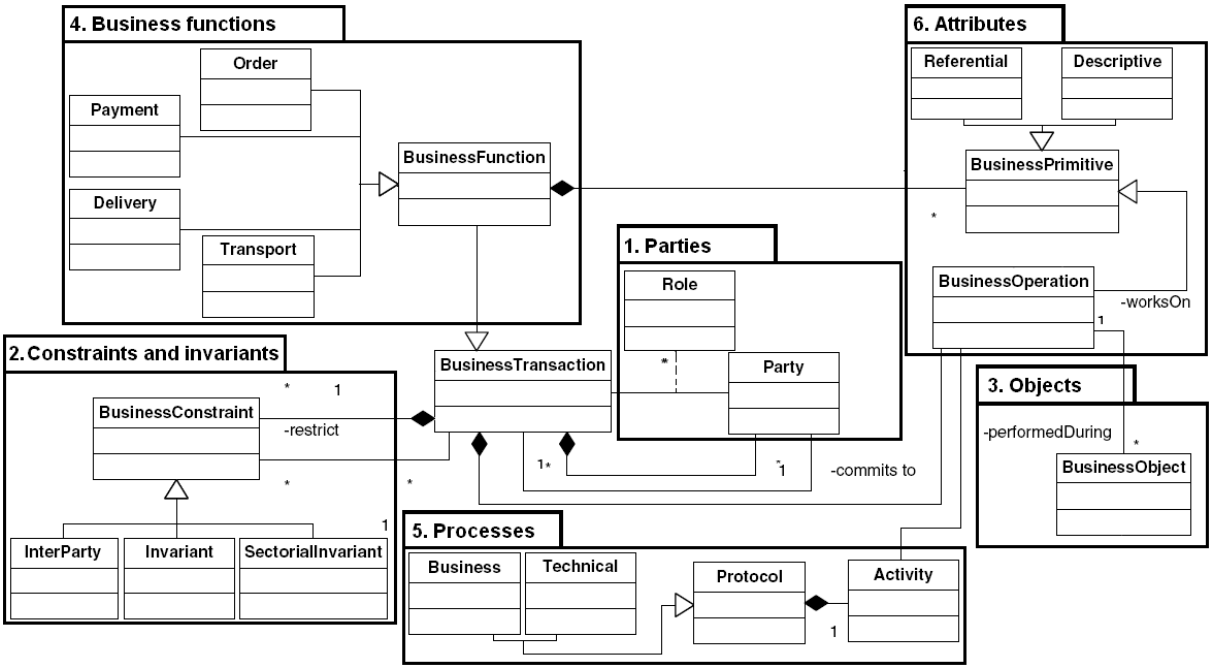


Fig. 2: Metamodel BTM in UML notation.

- Business operations are represented as methods in the roles.
- Business functions, i.e., business transactions by default, are represented by a parameterized model.
- Activities (are represented by icons of collaboration in which a partnership breaks down higher level of abstraction.
- BusinessPrimitive are divided into two types: first, referential primitive are represented as UML dependencies between classes, on the other hand, descriptive primitive are represented by enums of UML.

Figure 3 represents a business transaction using the proposed notation. The purpose of the transaction is to provide a travel plan according to the requirements of the traveler. This business transaction has a constraint, which that the airline to provide transportation service must be exclusively of low cost.

3.3 Notation for dynamic models of BTM

Figure 4, depicts the correlation between the meta-model of BTM and business processes using BPMN notation. The correlation used is as follows:

- The transaction is represented by a business process.
- The organizations that group the roles are represented by Pools.
- The roles of each organization are represented by Lanes.
- Activities in BTM are represented as activities in BPMN.
- Business objects are represented as DataObjects.

4. Transformation of static models to BPMN

As we explained above, BTM proposed the use of procedures to detail transactions. Since the transaction between the models and there is a correlation process described above, we have developed two ATL transformations which can obtain part of a partnership based on a UML BPMN and vice versa.

For the example used, we have a business transaction that creates a business process. The roles Airline, Traveler and Travel Agency was transformed into three Lanes with the same name. The three activities of BTM (travel search, evaluate and provide ticket preferences) are transformed to three BPMN activities.

5. Implementation

Through technological infrastructure we have developed an initial prototype to validate the feasibility of obtaining an executable prototype using the proposed models at CIM.

To achieve this, we first developed the infrastructure necessary to support models and transformations. For the static model of the transaction we developed a model editor using the Eclipse GMF tool. For modeling of business processes we have used a Eclipse plugin that allows BPMN modeling. Finally, and using the ATL transformation language, we develop two transformations to obtain a BPMN model from a model BTM and vice versa.

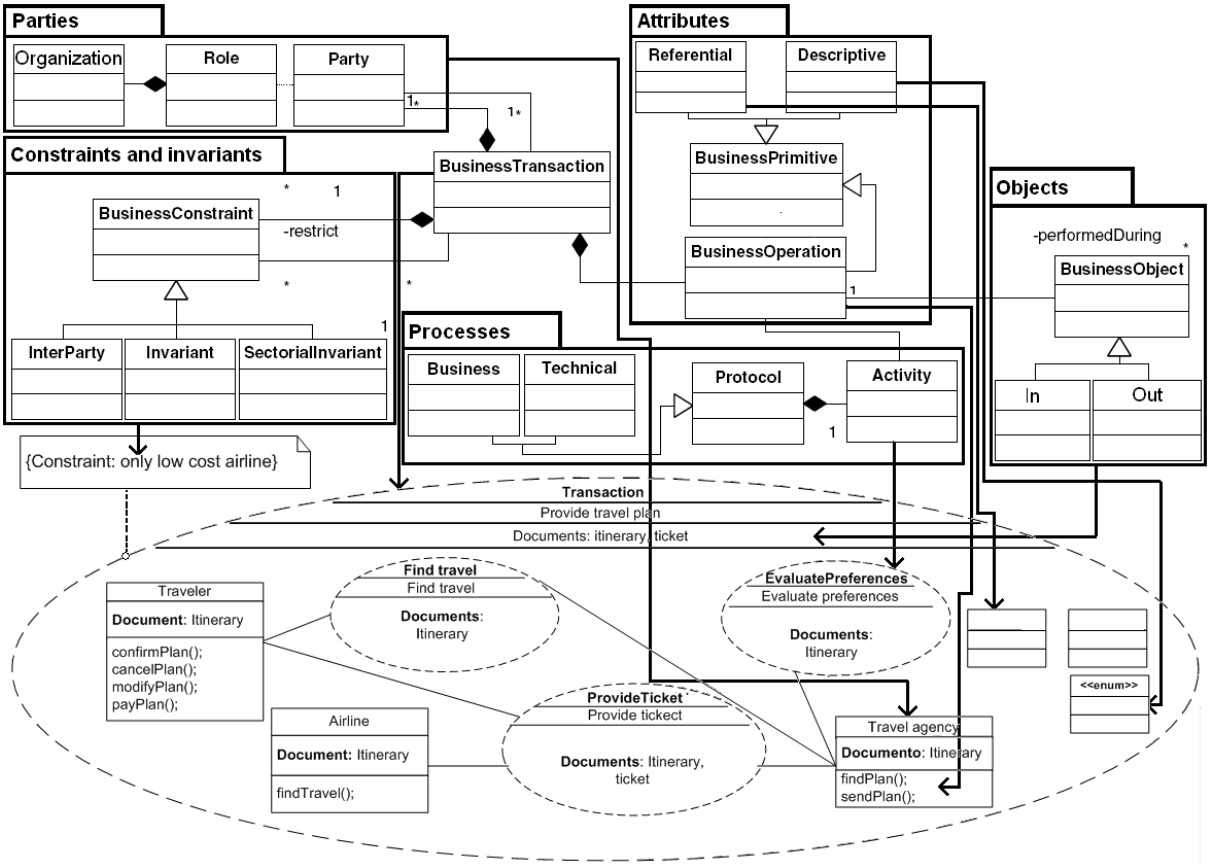


Fig. 3: Notation: static model.

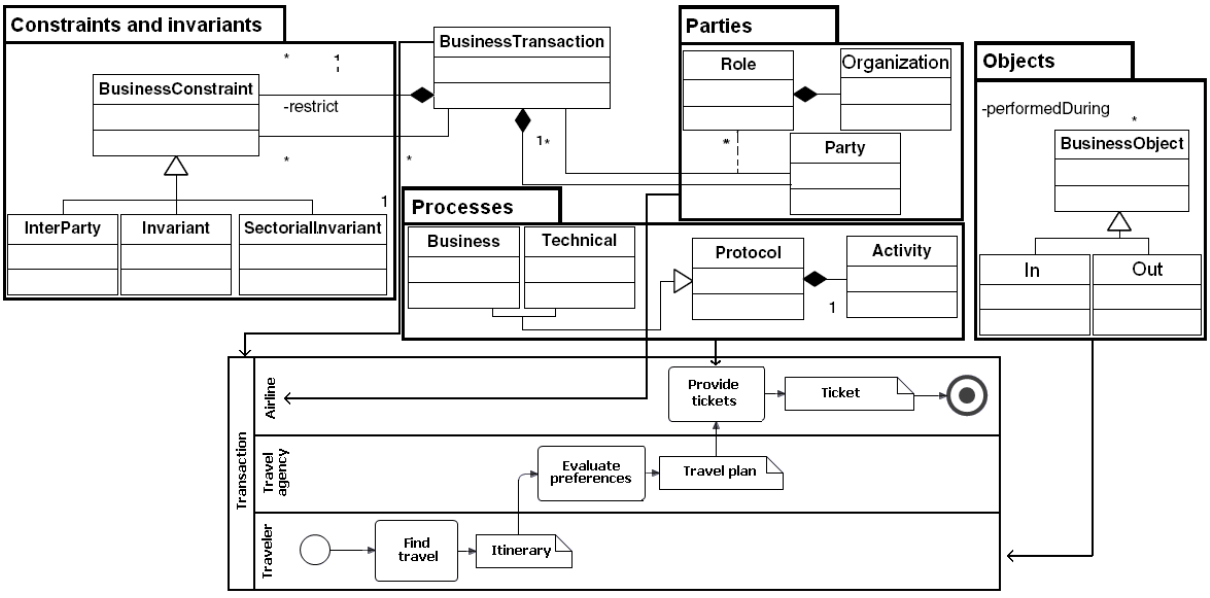


Fig. 4: Notation: dynamic model.

6. Conclusions

In this paper we have provided a meta-model adapted to MDD, a notation based on UML2 collaborations and a set of extensions to the instantiation of a business transaction model (BTM). We have developed an MDD transformation allows mapping some characteristics of a model BTM to business processes using BPMN notation and vice versa. Finally, in order to validate both the model and the transformation we have developed a prototype using the ATL transformation language, script and tool MoF Eclipse GMF.

7. Aknowledgements

This work has been supported by Universidad de la Amazonia, Fundación Carolina and by the Comité Departamental de Ciencia y Tecnología (CODECYT).

References

[1] Ibrahim, I., Schwinger, W., Weippl, E., Altmann, J., Winiwarter, W.: Agent Solutions for E-business Transactions, Database and Expert Systems Applications, Proceedings. 12th International Workshop on Volume , Issue , Page(s):84 - 87. (2001)

[2] Lee, H.: The Triple-A supply chain. Harvard Business Review, 82(10):102-112. (2004)

[3] López, G., De Castro, V., Marcos, E.: Implementation of Business Process Requiring User Interaction. OTM Workshops, LNCS 4277, pp. 107-115. (2006)

[4] Papazoglou, M., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing Research Roadmap. Technical report/vision paper on Service oriented computing European Union Information Society Technologies (IST), Directorate D - Software Technologies (ST). (2006).

[5] Papazoglou, M., Kratz, B.: A Business-Aware Web Services Transaction Model. In: ICSOC. (2006)

[6] Peña, J.: On improving the modelling of complex acquitance organisations of agents: A method fragment for the analysis phase. Ph.D thesis, Departamento de Lenguajes y Sistemas Informáticos. Universidad de Sevilla. (2005)

[7] Peña, J; Dominguez-Machuca, J. A.; Gonzalez-Zamora, M. M. A Roadmap For Future Research On The Specification Of Business Services In Supply Chain Management: The Quest For Synergy Between Software Engineering And Service Operations Management Fields Manufacturing Fundamentals: Necessity And Sufficiency (Proceedings of 3rd World Conference On Pom, Pom Tokyo 2008)

[8] Peña, J; Dominguez-Machuca, J. A.; Gonzalez-Zamora, M. M. Towards Specifying Instrumental Business Services In Supply Chain: A Preliminary Proposal. Actas Del XVII Congreso Nacional De Acede, Spain, 2007.

[9] Zinnikus, I., Benguria, G., Elvesæter B., Fischer, K., Vayssiere, J.: A Model Driven Approach to Agent-Based Service-Oriented Architectures. K. Fischer et al. (Eds.): MATES 2006, LNAI 4196, pp. 110-122. (2006)

SESSION

SOFTWARE QUALITY

Chair(s)

TBA

Coupling Detection to Facilitate Maintenance of Database Applications

Paul L. Bergstein and Ashwin Buchipudi
Dept. of Computer and Information Science, Univ. of Massachusetts Dartmouth, Dartmouth, MA, USA

Abstract – *Enterprise applications typically include a relational database layer. Unfortunately, the current generation of IDE's (Integrated Development Environments) do not adequately capture the interaction between the database management system and other layers of the application. For example, current Java IDE's do not evaluate the relationship of classes with the database, or how a particular java method interacts with database tables and columns. We report here our recent progress in developing an Eclipse plug-in that helps the programmer by providing a visual map of interactions between Java code and relational databases. A primary motivation is to facilitate code maintenance in the face of database modifications.*

Keywords: Software maintenance, software visualization tools.

1 Introduction

Modern tools have simplified the development of enterprise applications by bridging gaps across various technologies like file systems, relational databases, messaging, and web services. However, this has also led to challenges in maintenance and enhancement of enterprise applications. An enterprise application usually consists of a web layer, the business logic and relational database, often enhanced with frameworks like Struts and Hibernate for web and persistence. However, the interaction between these various layers is not sufficiently captured by the current generation of IDE (Integrated Development Environment). For example, the Eclipse IDE provides support for syntax and debugging of java classes, but it does not evaluate the relationship with the database, or how a particular java method interacts with database tables and columns. For example, it does not flag a warning where an SQL query might be formed incorrectly. Similarly, the Visual Studio .NET would not flag a warning if an XPath applied on an XML document does not correspond to a valid value according to the schema. This makes it very difficult to maintain and enhance applications written by a third party, since a change in code may break some other layer, and the problem will become known only after extensive testing.

Our goal is to develop a framework that will help programmers in bridging the gap between different technologies used in an enterprise application. However, this is very substantial initiative and we report here our progress in developing an Eclipse plug-in that helps the programmer by providing a visual map of interactions between Java code and relational databases.

The obvious benefit of the mapping is to facilitate code maintenance in the face of database modifications by identifying the code-to-database couplings. In some cases the string search function of the IDE's editor might be useful for finding affected code when changes are made to the database schema. However, this technique is difficult or impossible to use in certain situations. Suppose, for example, that a column name is changed in one table, but other tables have columns with the same name. A search for the column name in the java code may find many instances that are irrelevant. Furthermore, consider that table and column names may be stored in variables, passed as parameters to other methods, or constructed dynamically in code (e.g. by string concatenation). In such cases, the string search technique may fail to detect many areas of affected code. With our tool, the developer only needs to click on a database element to find the code coupled to that element.

A second, and equally important, benefit involves the easy detection of code-to-code couplings that arise when different java methods access the same database elements. Suppose a developer has a Java enterprise application which uses relational database for data persistence, and the Eclipse IDE is being used for

development. The programmer wants to make some changes to a method and would like to know the effects of this change on rest of the code. Ordinarily the programmer could use the "Call Hierarchy" feature of the Eclipse IDE to get the dependencies of other methods and classes on this method. But suppose the method uses an SQL statement to store a string in the *address* column of the *customer* table, and the developer wants to change the format of the address. This is not easy because there may be many other methods which are dependent on the address format but are not related to the current method containing this SQL query through the call hierarchy. As noted above, the editor's search function is not a reliable way to find the affected methods. Therefore the programmer has to manually inspect all the classes and check for methods referencing the address column of the customer table, but even this manual process is highly error prone when column and table names are passed as parameters and accessed through parameter or variable names.

2 Background

We have previously reported [1,2] our development of a prototype tool to provide a visual mapping of java code-to-database couplings. In our initial effort we developed a stand alone application to scan java source code and present the coupling information to the user in tabular format. After entering a java source file name and database connection information (database, host, username, and password) the tool would scan the source code for database access and display tables with the couplings that were found.

When we tested the prototype on a simple database application development project, we found it to be quite useful, but several shortcomings were apparent. First, our methodology for identifying the code-to-database couplings depended entirely on static analysis of the java source code, and was not very sophisticated. Second, we had no mechanism for tracking changes to the code and database schema over time, so that the developer would need to identify the relevant couplings *before* making a change. For example, if the developer were to change the name of a database column, it would be necessary to find the couplings of code to that column before the change was made. Afterward, the couplings would no longer exist and our tool had no way to identify the affected code.

Also, our user interface was a bit awkward, not integrated into a development environment, and only allowed the user to process a single source file at a time. Therefore it was not very useful for detecting code-to-code couplings between different source files.

In our more recent implementations we have made improvements to address each of these shortcomings. We have implemented our tool as a fully integrated plug-in to the popular Eclipse development environment with the ability to visualize all the code-to-database and code-to-code (via database) couplings for an entire project. We have continued to improve our static code analysis and added a dynamic (runtime) analysis feature. We have added change tracking ability by storing the coupling information in a database that the tool uses internally.

The current implementation also enhances the user interface by providing the ability to view the database and the project at various levels of granularity. For example, users can choose to view couplings of code to anywhere in the database, to a particular table in the database, or to a specific column in a table. Similarly, they can adjust the granularity of their project view between the project, class, and method levels.

Search facilities have also been enhanced to enable users to easily find the types of coupling they are most interested in. For example, when a method that stores information in the database is modified, it is easy to find all of the methods (or classes or projects) that retrieve the same information and might be affected.

3 Results

Figure 1 shows the overall architecture of the tool. The tool uses both static and dynamic analysis of the java code to find database couplings. The results of both analysis methods are combined in the coupling data repository which is also used to track changes over time. The user interface, implemented as an Eclipse plug-in displays the results to the user and allows easy navigation to code based on its database coupling.

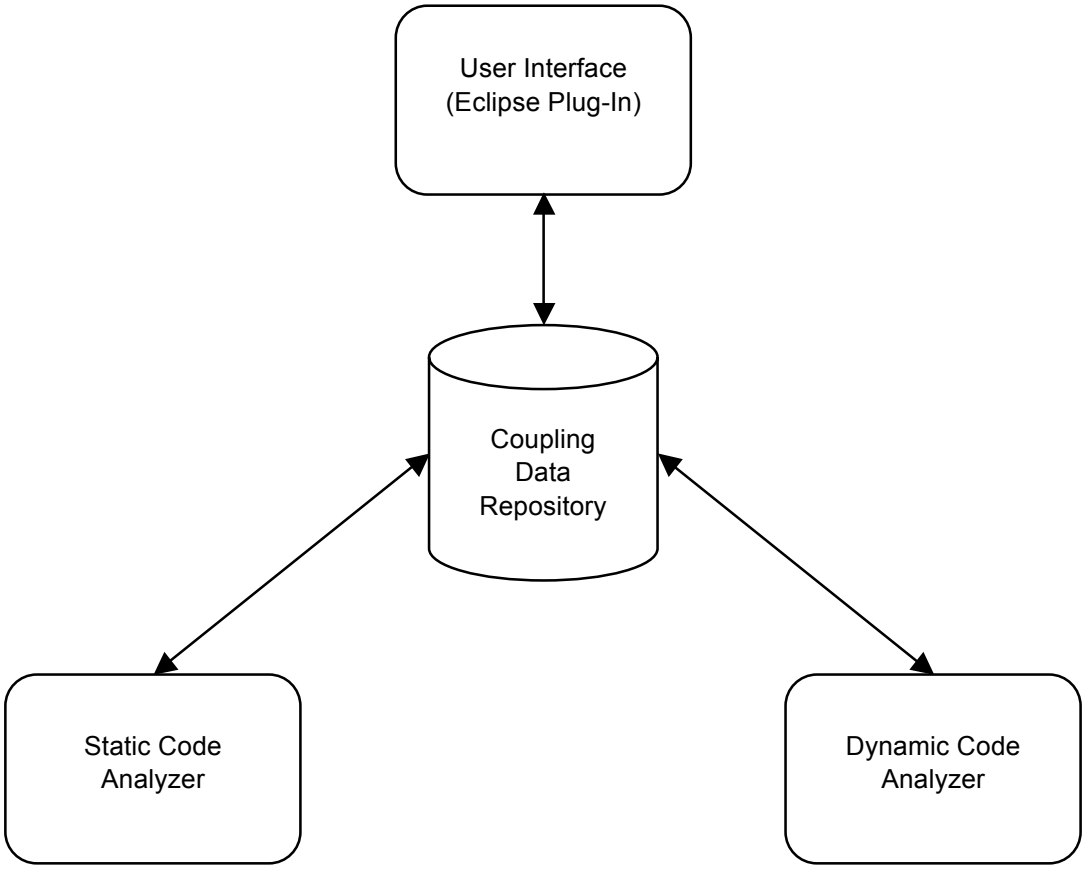


Figure 1

3.1 Static Code Analysis

The static code analyzer uses the Sun java compiler API [11] and the Compiler Tree API [12] to parse the java source and walk the abstract syntax tree. The static code analyzer performs its own parsing, so that it can identify incomplete SQL fragments as well as complete valid SQL statements. In particular, it looks for string literals that are included either directly or after assignment to String variables in calls to the *execute*, *executeQuery*, and *executeUpdate* methods of the JDBC *Statement* class. The analyzer attempts to identify column and table names occurring in select, from and where clauses and record these dependencies in the coupling repository.

Although this ability is still somewhat primitive, the code analyzer is able to detect simple cases of dynamic SQL generation in the code. The analyzer considers certain string concatenations including some concatenations that are built from a combination of string literals and variables.

However, static analysis in general is a hard problem and it will never be possible to detect all couplings to the database that may occur at runtime, possibly dependent on user input. We have therefore focused most of our recent code analysis work on dynamic techniques.

3.2 Dynamic Code Analysis

In order to overcome some of the difficulties of static code analysis, we have implemented dynamic code analysis in our system. The main component of the dynamic analyzer is a JDBC bridge driver that logs the database accesses to the coupling data repository. Our driver acts as a bridge between the application and

the "real" driver that communicates with the user's database. The implementation is conceptually simple. Most of the methods in our driver classes simply pass requests on to the underlying "real" driver and return whatever data is returned from the real driver. The main exception is in the Statement class methods (e.g. execute, executeQuery, executeUpdate) that take SQL statements as arguments. These methods receive only complete, valid, fully formed SQL statements as arguments (unless there are errors in the application) even if they have been built dynamically.

The SQL statements processed in the JDBC driver are easily parsed with the ZQL [10] SQL parser to determine the database elements that are being accessed. The driver methods that process the SQL statements create (but don't throw) Exceptions and use the Exception object to obtain a stack trace. Using methods in the StackTraceElement class, the driver can determine the class, method, file, and line number from which it was called. The coupling information is recorded in the coupling data repository.

In order to ensure that all JDBC database access goes through our bridge driver, we also supply a replacement for the DriverManager class. Installation of the dynamic analyzer requires installation of the bridge driver and replacing the standard DriverManager.

3.3 Coupling Database

The coupling data repository is implemented as a database that is used internally by our tool. For every code-to-database coupling that is detected by either the static or dynamic code analyzer, there is an entry in the repository. Each coupling entry in the repository includes the code location (class, method, file, and line number), the database element (database, table, and column), the SQL statement type (select, insert, update, etc.) and the type of access (read, write, or read/write). The statement type does not necessarily determine the access type. For example, a field occurring in the *set* clause of an update statement indicates a write access, but a field occurring in the *where* clause of the same statement indicates a read access.

In order to detect changes over time, the repository also records the first time and last time that a coupling is detected. Also, each time the tool is run, the structure of the database is checked using the JDBC metadata API, and any structural changes are recorded in the repository.

3.4 User Interface

The screenshot in Figure 2 shows the tool interface in an Eclipse pane. On the Database View tab the database structure is shown as a tree with database, table, and column information arranged in a hierarchal structure. Selecting an element from this tree brings the associated couplings into view along with the controls to select sorting options. In the example in Figure 2, the tree is collapsed to a single node (which is selected), and all couplings to the database are displayed. Selecting a code reference from the coupling list brings the corresponding java source into view in an editor pane with the appropriate line of code highlighted.

When two or more methods are coupled to the same database element, there is a suggestion that these methods may be coupled through the database. The nature of the coupling can be seen from the statement type and access type information. For example, one method might read data that is written to the database by another method.

The interface also allows the user to browse in the opposite direction, i.e. from code to database. The project view tab provides a hierarchal view of the projects where the user can select a project, class, or method to find the database elements that it accesses. A developer can find methods that are coupled through the database to a method that has been added or modified by using both views. First, the database elements accessed by the new or modified method can be found in the project view tab. Then, other methods that access the same database elements can be found in the database view tab. In future versions, we plan to improve the interface to automate this task.

4 Related Work

There is a large body of work on software visualization [3-7] and also on database visualization. There is also a good deal of work on reverse engineering of databases and CASE tools that support reverse engineering with visualization techniques. However, we are not aware of any other system designed to support the development and maintenance of software through the visualization of program code

dependencies on the database.

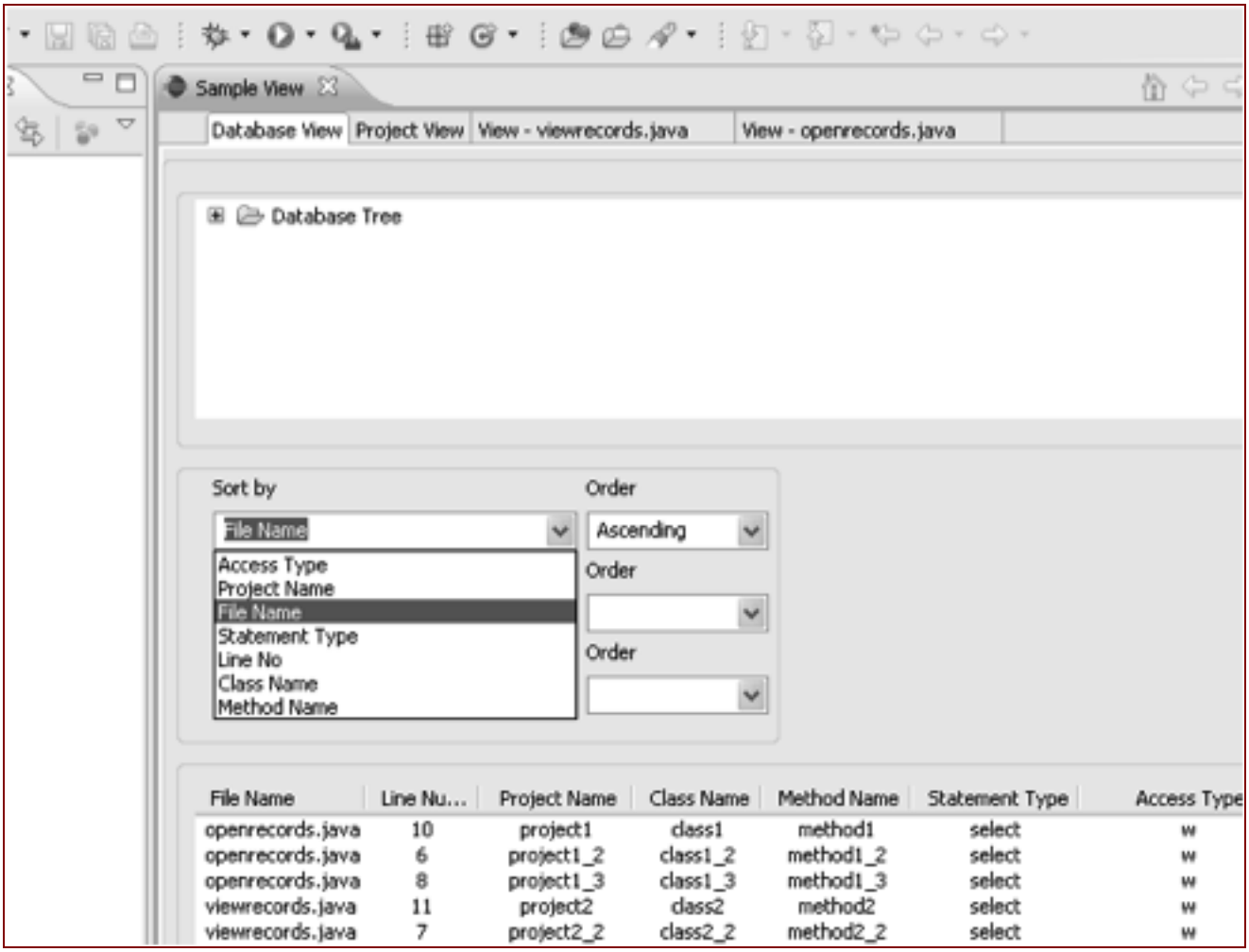


Figure 2

5 Conclusions

Many researchers have investigated to resolve the dependencies between different technologies involved in an enterprise application. Our tool significantly enhances visibility between java and relational databases. The principal benefit is the ability to easily detect the code-to-database couplings and couplings of code-to-code via the database. This ability makes it easy to maintain application code in the face of structural changes to the database, or changes in the format of data stored in the database.

Static and dynamic analysis of java code to discover database couplings each have their advantages and disadvantages. Dynamic analysis is easier to implement and will find all couplings that occur during testing. Static analysis is harder to implement and cannot identify couplings that only occur dynamically (e.g. based on user input). However, static analysis may identify couplings that are missed during the testing phase. By combining the results of static and dynamic analysis in a coupling data repository, we get the combined benefits of each. The repository also allows for tracking of changes over time so that areas

of code that may be affected by a change could be flagged for the developer.

6 Future Work

We test our tool on medium sized applications that are developed as team based student projects in a database course. So far, our users have found the tool to be very useful. However, we are actively working on improvements to the user interface based on their feedback. In particular, our users are most interested in automating the discovery of code-to-code couplings through the database, and in automatic flagging of potentially affected code when structural changes to the database occur. We are also working on improving the static code analyzer to reduce the number of couplings that are only detected dynamically.

In the long term, we plan to extend this tool to handle additional languages and technologies. For example, we plan to extend our java code analyzers to support JSP by analyzing the java snippets embedded in JSP pages, so that we can show couplings of JSP pages to the database. This would also allow the visualization of couplings between the presentation layer (JSP) and business logic code that occur through the database in a typical J2EE environment. If all these dependencies between the various layers of a J2EE application can be shown through a visual tool, the task of maintaining and enhancing such applications would be greatly facilitated. Eventually, we would also like to support additional programming languages such as C# and C++ and add support for ODBC applications.

7 References

[1] Sai Ravindran and Paul L. Bergstein. AppDetector: A Tool Prototype for Visualizing Java Code Dependencies on Relational Databases. In *Proceedings of the 2007 International Conference on Software Engineering Research and Practice (SERP'07)*, Pages 497-500, June 25-28, 2007, Las Vegas, Nevada. CSREA Press, ISBN 1-60132-034-5.

[2] Paul L. Bergstein, Priyanka Gariba, Vaibhavi Pisolkar, and Sheetal Subbanwad. An Eclipse Plug-In for Visualizing Java Code Dependencies on Relational Databases. In *Proceedings of the 2009 International Conference on Software Engineering Research and Practice (SERP'09)*, Pages 64-69, July 13-16, 2009, Las Vegas, Nevada. CSREA Press ISBN 1-60132-129-5.

[3] G. C. Roman and K. C. Cox. A taxonomy of program visualization systems. *IEEE Computer*, Vol. 26(12), Pages 11-24, 1993.

[4] Blaine A. Price, Ian S. Small, and Ronald M. Baecker. A Principled Taxonomy of Software Visualization. *Journal of Visual Languages and Computing*, Vol. 4, Pages 211-266, 1993.

[5] Jonathan I. Maletic, Andrian Marcus, and Michael L. Collard. A task oriented view of software visualization. In *Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT)*, Pages 32-40, 2002.

[6] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, Pages 77-86, 2003. ACM Press.

[7] M. D. Storey, K. Wong, F. D. Fracchia, and H. A. Müller. On Integrating Visualization Techniques for Effective Software Exploration. In *Proceedings of IEEE Symposium on Information Visualization*, Pages 38-45, 1997.

[8] Terence Parr. An Introduction To ANTLR. http://www.cs.usfca.edu/~parrt/course/652/lectures/antlr.html

[9] Java Grammar http://www.antlr.org/grammar/java

[10] Zql: a Java SQL parser http://www.experlog.com/gibello/zql/

[11] Java 6 Compiler API http://today.java.net/pub/a/today/2008/04/10/source-code-analysis-using-java-6-compiler-apis.html

[12] Compiler Tree API http://java.sun.com/javase/6/docs/jdk/api/javac/tree/index.html

Functional-Object-Oriented Hybrid Programming with FOBS

James Gil de Lamsdrid
Bowie State University
Bowie, MD 20715

Jill Zimmerman
Goucher College
Baltimore, MD 21204

Keywords: hybrid, functional, object-oriented, languages

Abstract:

We describe a computer language that is a hybrid between functional and object-oriented languages. The FOBS interpreter consists of a macro processor that translates an extended FOBS language into a core FOBS language. In this paper we restrict our discussion to the core FOBS language. The language is based on a simple structure called a FOB (functional-object), capable of being used as a function, or accessed as an object. FOBS is a dynamically typed interpreted language, designed for use as a universal scripting language. An extensive library is integral to the language. The library implements the primitive types and provides an interface to the external environment, allowing scripting actions to be carried out.

Introduction

The object-oriented programming paradigm and the functional paradigm both offer valuable tools to the programmer. Many problems lend themselves to elegant functional solutions. Others are better expressed in terms of communicating objects. FOBS is a single language with the expressive power of both paradigms allowing the user to tackle both types of problems, with fluency in only one language.

FOBS has a distinctly functional flavor. In particular, it is characterized by the following features:

- A single, simple, elegant data type called a FOB, that functions both as a function and an object.
- A stateless runtime environment. In this environment, mutable objects are not allowed. Mutation is accomplished, as in functional languages, by the creation of new objects with the required changes.
- A simple form of inheritance. A *sub-FOB* is built from another *super-FOB*, inheriting all attributes from the super-FOB in the process.
- A form of dynamic scoping to support attribute overriding in inheritance. This allows a sub-FOB to replace data or behaviors inherited from a super-FOB.

As with many scripting language FOBS is weakly typed, a condition necessitated by the fact that it only has one data type. However, with interpreted languages the line between parsing and execution is more blurred than with compiled languages, and the necessity to perform extensive type checking before execution becomes less important.

Several researchers have built hybrid language systems, in an attempt to combine the functional and object-oriented paradigms, but have sacrificed referential transparency in the process. Yau et al. [1] present a language called PROOF. PROOF tries to fit objects into the functional paradigm with little modification to take into account the functional programming style. The language D by Alexandrescu [2] is a rework of the language C transforming it into a more natural scripting language similar to Ruby and Javascript.

Two languages that seek to preserve functional features are FLC by Beaven et al. [3], and FOOPS by Goguen and Mesegner [4]. FOOPS is built around the addition of ADTs to functional features. We feel that the approach of FLC is much cleaner. In FLC, classes are represented as functions. This is the basis for FOBS also. In FOBS we have, however, removed the concept of the class. In a stateless environment, the job of the class as a "factory" of individual objects, each with their own state, is not applicable. In stateless systems a class of similar objects is better represented as a single prototype object that can be copied with slight modifications to produce variants.

Language Description

FOBS has only one type of data: the FOB. A *simple FOB* is a quadruplet,

[*m i* -> *e* ^ ρ]

The FOB has two tasks. Its first task is to bind an identifier, *i*, to an expression, *e*. The *e-expression* is unevaluated until the identifier is accessed. Its second task is to supply a return value when invoked as a function. ρ (the ρ-*expression*) is an unevaluated expression that is evaluated and returned upon invocation.

The FOB also includes a modifier, *m*. This modifier indicates the visibility of the identifier. The possible values are: "`+", indicating public access, "`~", indicating protected access, and "`$", indicating argument access. Identifiers that are protected are visible only in the FOB, or any FOB inheriting from it. An argument identifier is one that will be used as a formal argument, when the FOB is invoked as a function. All argument identifiers are also accessible as public.

For example, the FOB

```
[`+x -> 3 ^ 6]
```

is a FOB that binds the variable *x* to the value 3. The variable *x* is considered to be public, and if the FOB is used as a function, it will return the value 6.

Primitive data is defined in the FOBS library. The types *Boolean*, *Char, Real*, and *String* have constants with forms close to their equivalent *C* types. The *Vector* type is a container type, with constants with form close to that of the ML list. For example, the vector

```
["abc", 3, true]
```

represents an ordered list of a string, an integer, and a Boolean value. Semantically, a vector is more like the *Java* type of the same name. It can be accessed as a standard list, using the usual *car*, *cdr*, and *cons* operations, or as an array using indexes. Unlike the *Java* type, the FOBS type is immutable. The best approximation to the mutate operation is the creation of a brand new modified vector.

There are three operations that can be performed on any FOB. These are called *access*, *invoke*, and *sequence*.

An access operation accesses a variable inside a FOB, provided that the variable has been given a public or argument modifier. As an example, in the expression

```
[`+x -> 3 ^ 6].x
```

the operator "." indicates an access, and is followed by the identifier being accessed. The expression would evaluate to the value of *x*, which is 3.

An invoke operation invokes a FOB as a function, and is indicated by writing two adjacent FOBs. In the following example

```
[`$y -> _ ^ y.+[1]] [3]
```

a FOB is defined that binds the variable *y* to the empty FOB and returns the result of the expression *y* + 1, when used as a function. To do the addition, *y* is accessed for the FOB bound to the identifier "+", and this FOB is invoked with 1 as its actual argument. The full FOB defining *y* is then invoked, passing as actual argument the value 3. The result of the invocation is 4.

In invocation, it is assumed that the second operand is a vector. This explains why the second operand in the above example is enclosed in square braces. Invocation involves binding the actual argument to the argument variable in the FOB, and then evaluating the ρ–expression, giving the return value.

A sequence operation is indicated with the operator ";". It is used to implement inheritance. In the following example

```
[`+x -> 3 ^ _] ;
[`$y -> _ ^ x.+[y]]                (1)
```

two FOBs are sequenced. The *super-FOB* defines a public variable *x*. The *sub-FOB* defines an argument variable *y*, and a ρ-expression. Notice that the sub-FOB has unrestricted access to the super-FOB, and is allowed access to the variable *x*, whether modified as public, argument or protected.

The FOB resulting from Expression (1) can be accessed, invoked, or further sequenced. For example the code

```
([`+x -> 3 ^ _] ;
[`$y -> _ ^ x.+[y]]).x
```

evaluates to 3, and the code

```
([`+x -> 3 ^ _] ;
[`$y -> _ ^ x.+[y]]) [5]
```

evaluates to 8.

Multiple sequence operations result in *FOB stacks*, which are compound FOBs. For example, the following code creates a FOB with an attribute *x* and a two argument function that multiplies its arguments together. The code then uses the FOB to multiply 9 by 2.

```
([`+x -> 5 ^ _] ; [`$a -> _ ^ _] ;
[`$b -> _ ^ a.*[b]]) [9, 2]
```

In the invocation, the arguments are substituted in the order from top to bottom of the FOB stack, so that the formal argument *a* would be bound to the actual argument 2, and the formal argument *b* would be bound to 9.

In addition to the three FOBS operations, many operations on primitive data are defined in the FOBS library. These operations include the usual arithmetic, logic, and string manipulation operations. In addition, conversion functions provide conversion from one primitive type to another, when appropriate.

We present a larger example to demonstrate how FOBS code might be used to solve more complex programming problems. In this example we build the binary search tree, shown in Fig. 1, and then search it for the character 'f'. In the FOBS solution, we construct a FOB with the structure shown in Fig. 2. The *Node* FOB is the prototype that is copied to create *Node* objects. The method called *r.v.* indicates the return value of the FOB.

The FOBS code for the example follows.

```
## definition of the NodeMaker FOB
([NodeMaker ->
      [`$lt -> _ ^ _] ;
      [`$rt -> _ ^ _] ;
      [`$in -> _ ^ _] ;
      [`~Node ->
         [`~left -> lt ^ _] ;
         [`~right -> rt ^ _] ;
            [`~info -> in ^ _] ;
            [`~_ -> _ ^
               ([`~a1 -> info.=[key]
                  ^ _] ;
               [`~a2 -> left.=[_].|
                     [a1].if[false,
                left[key]] ^ _];
               [`~a3 -> right.=[_].|
                  [a1].if[false,
                  right[key]]^_];
               [`+a4 -> a1.|[a2].|[a3]
                  ^ _]).a4] ^ _]
         ^ Node] ;
## build the tree
[`+tree ->
      NodeMaker['m', NodeMaker['p', _,
   _], NodeMaker['g', NodeMaker['j',
   _, _], NodeMaker['f', _, _]]]
   ^ _])
## search for 'f'
.tree['f']
#.
```

This code has two types of elements: a FOB expression, and macro directives. Macro directives begin with the "#" character, and are expanded by the macro preprocessor. The two seen here are the comment directive, "##", and the *end of expression* directive, "#.".

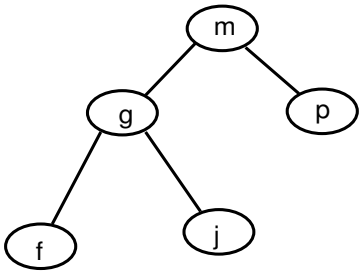


Fig. 1. Example binary search tree.

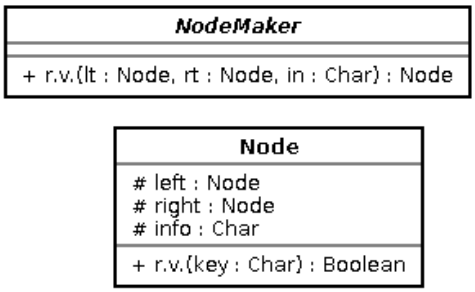


Fig. 2. Example FOB structure.

The top-level FOB defines a FOB *NodeMaker*, and the search tree, *tree*. The top-level FOB is accessed for the tree, and it is searched for the value 'f', using the invoke operator.

NodeMaker creates a FOB with the required attributes, and a return value that does a search. The return value uses the local variables *a1*, *a2*, *a3*, and *a4* to save the results of the comparison with the node, the left child, the right child, and the final result, respectively.

FOBS code is run by feeding FOBS expressions to the interpreter. Each FOBS expression produces as its value a single FOB, which is returned back to the user. This example would cause the value `true` to be printed.

Design Issues

Expression evaluation in FOBS is fairly straight forward. Three issues, however, need some clarification. These issues are: the semantics of the redefinition of a variable, the semantics of a FOB invocation, and the interaction between dynamic and static scoping.

Variable overriding

A FOB stack may contain several definitions of the same identifier, resulting in overriding. For example, in the following FOB

```
[`$m -> 'a' ^ m.toInt[]] ;
[`+m -> 3 ^ m]
```

the variable *m* has two definitions; in the super-FOB it is defined as an argument variable, and in the sub-FOB another definition is stacked on top with *m* defined as a public variable. The consequence of stacking on a new variable definition is that it completely overrides any definition of the same variable already in the FOB stack, including the modifier. In addition, the new return value becomes the return value of the full FOB stack.

Argument substitution

As mentioned earlier, the invoke operator creates bindings between formal and actual arguments, and then evaluates

the ρ-expression of the FOB being invoked. At this point we give a more detailed description of the process.

Consider the following FOB that adds together two arguments, and is being invoked with values 10 and 6.

```
([`$r -> 5 ^ _] ;
[`$s -> 3 ^ r.+[s]]) [10, 6]
```

The result of this invocation is the creation of the following FOB stack

```
[`$r -> 5 ^ _] ;
[`$s -> 3 ^ r.+[s]] ;
[`+r -> 6 ^ r.+[s]] ;
[`+s -> 10 ^ r.+[s]]
```

In this new FOB the formal arguments are now public variables bound to the actual arguments, and the return value of the invoked FOB has been copied up to the top of the FOB stack. The return value of the original FOB can now be computed easily with this new FOB by doing a standard evaluation of its ρ-expression, yielding a value of 10.

Variable scope, and expression evaluation

Scoping rules in FOBS are, by nature, more complex than scoping used in most functional languages. Newer functional languages, such as *Haskell* and ML, typically use lexical scoping. Dynamic scoping associated with older dialects of LISP, has recently fallen out of favor.

Pure lexical scoping does not cope well with variable overriding, as understood in the object-oriented sense, which typically involves dynamic message binding. To address this issue, FOBS uses a hybrid scoping system which combines lexical and dynamic scoping.

Consider the following FOB expression.

```
[`~u -> 1 ^ _] ;
[`+f ->
      [`~v -> 2 ^ u.+[v]]
^ _] ;
[`+g ->
      f ;
      [`~v -> 3 ^ u.+[v]]
^ _]                                    (2)
```

This expression defines a FOB stack that is three deep, containing declarations for a protected variable *u*, with value 1, and two simple public FOBs, *f* and *g*. Both *f* and *g* return the value of *u* + *v*. The variable *v* is defined in both the FOB *f* and the FOB *g*, with different values. The FOB *g* is a sub-FOB of the FOB *f*.

We are currently mostly interested in the FOB stack structure of Expression (2), and can represent it graphically with the *stack graph*, given in Fig. 2. In the stack graph each node represents a simple FOB, and is labeled with the variable defined in the FOB. Since there are two definitions of the variable *v*, they have been subscripted with the values to which they are bound in order to distinguish them from each other. Two types of edges are used to connect nodes: the *s-pointer*, and the *t-pointer*.

The s-pointer describes the lexical nested block structure of one FOB defined inside of another. The s-pointer for each node points to the FOB in which it is defined. For example v_2 is defined inside of the FOB *f*.

The t-pointer for each node points to the super-FOB of a FOB. It describes the FOB stack structure of the graph. In Fig. 3 there are basically two stacks: the top level stack consists of nodes *u*, *f*, and *g*, and the nested stack consisting of nodes v_2, and v_3.

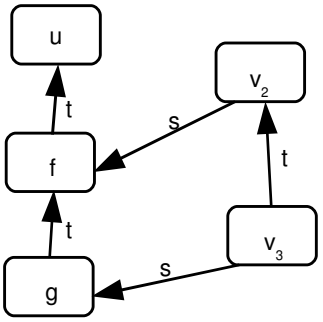


Fig. 3. Stack graph of Expression (2).

If the FOB *f* were invoked, the value of *u*, 1, would be added to the value of v_2, 2, yielding a value of 3. The fact that *u* is visible from v_2 is a result of the lexical scoping used in FOBS, that follows the s-pointers outward.

Instead of *f*, if the FOB *g* were invoked, the value of *u*, 1, would be added to the value of v_3, 3, yielding a value of 4. In this situation, *g* has two definitions of the variable *v*: v_2, which is the lowest in the FOB stack, and v_3, which is the higher in the stack. The higher definition overrides the lower definition.

Notice that which value of *v* is used, depends on the *entry point* into the nested stack. If the stack is entered through *f*, v_2 is used, and if it is entered through *g*, v_3 is used. This introduces a dynamic element into the scoping rules in which FOB stacks are searched using the t-pointers, starting from the entry point, down to the bottom of the stack.

On occasion, the two scoping rules may conflict, and so it is necessary to have a precedence rule to resolve such conflicts. Take for example the following FOBS expression.

```
[`~u ->  1 ^ _] ;
[`~f ->
      [`~u ->  2 ^ _];
      [`~_ -> _ ^ u]
^ _]
```

The FOB *f* returns *u*, but there are two definitions of *u* to choose from: u_1 is from the outside FOB, and u_2 is from down in the stack. In this case dynamic scoping takes precedence, and so if *f* were invoked, the value of u_2 would be returned.

Conclusion

We have presented a core FOBS language. This language is designed as the basis of a universal scripting language. It has a simple syntax and semantics.

The language described is the core of an extended language in which programs will be translated into the core by a macro processor. This will allow for a language with syntactic "sugar", that still has the simple semantics of our core FOBS language.

FOBS is a hybrid language, which combines the tools and features of object oriented languages with the tools and features of functional languages. In fact, the defining data structure of FOBS is a combination of an object and a function. The language provides the advantages of referential transparency, as well as the ability to easily build structures that encapsulate data and behavior. This provides the user the choice of paradigms.

References

[1] Yau, S.S., Jia, X., Bae, D. H., *Proof: A Parallel Object-Oriented Functional Computation Mode*l, Journal of Parallel Distributed Computing, 12, 1991.

[2] Alexandrescu, A., *The D Programming Language*, Adison Wesley, 2010.

[3] Beaven, M., Stansifer, R., Wetlow, D., *A Functional Language with Classes*, Lecture Notices in Computer Science, 507, 1991.

[4] Goguen, J. A., Mesegner, J., *Unifying Functional, Object-Oriented, and Relational Programming with Logical Semantics*, Research Directions in Object-Oriented Programming, MIT Press, 1987, pp. 417-478.

Secure Reliability of Measurement Data through Application of Mechanism Design Theory

Sang-Pok Ko, Woo-Bok Yi
Memory Division, Semiconductor Business, Samsung Electronics Co., Ltd
San #16 Banwol-Dong, Hwasung-City, Gyeonggi-Do, South Korea
{sangpok.ko, woobok yi}@samsung.com

Abstract

Recently, many corporations have applied CMM, SPICE or other software process maturity models to developing software products. These models are recommended to change from qualitative process management to quantitative management as they reach a certain level of maturity. Quantitative process management is possible only when reliable data exist. If quantitative process management are conducted when data's reliability are not ensured, adverse effects can be caused by the false management activities. That is why so many methods have been studied to analyze reliability of collected data. Most of them, however, are analysis methods for already collected data. Therefore, if the analysis finds that the existing data are not reliable, the data should be abandoned and new data be collected again, causing need to consume much time and effort. Therefore, as a way to secure reliability of data to be used for quantitative process management when they are input in the first place, this paper suggests a method applying the second level meta game of non-cooperative, bimatrix games to collecting data for quantitative measurement.

Keywords : S/W Measurement, S/W Metrics, S/W Quality, Game theory, Quantitative management.

1. Introduction

One object of software engineering is "producing high-quality products at minimum cost, within a short period." [1]. To this end, many methods and tools, models have been developed. Among them are CMM(Capability Maturity Model)[2] of SEI(Software Engineering Institute) and SPICE(Software Process Improvement & Capability determination)[3] of ISO/IEC 15504 TR2(Technical Report Type 2). Those models recommend that, if maturity of process reaches a certain level, metrics of development be quantitatively measured and used for quantitative management.

Quantitative management is a technique to perform a task by representing performance data in numbers and setting the quantitative goals based on the numbers. In quantitative management, most important is reliability of collected data. If reliability of the data is not guaranteed, the quantitative management based on the data is also meaningless.

Many methods have been studied to analyze reliability of collected data. Most of them, however, are just analysis methods for already collected data. Therefore, this paper suggests a technique applying a game theory to collecting input data as a way to secure reliability of data to be collected.

This paper consists of following parts: Chapter2 outlines of measure, game theory in S/W engineering, and business psychology; Chapter3 reviews obstacles to data collecting and analysis of their weight values; Chapter4 deals with the technique applying a game theory to collecting data to be measured in order to get reliable data; Chapter5 demonstrates the technique described in the forth chapter through simulation; finally, Chapter 6 concludes this paper and provides study direction for the future.

2. Related Research

2.1 Measure in Software Engineering

In software engineering, measure is used for three purposes: for software process improvement; for project

management involving estimation, quality management, productivity evaluation, and project control; as a data to make a technical decision when implementing a project.

Lord Kelvin once said, " When you can measure what you are speaking about, and express it in numbers, you know something about it. But when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind." [4] This means that software is hard to manage the development process, compared to hardware, because software development process, in much part, is highly dependant on people and thus, the measure is mainly qualitative. But if managed by quantitative measure, it can be managed in effective and systematic way.

2.2 Software Process Model

CMM and SPICE represent software process models. CMM consists of 5 maturity levels from 1 to 5: level 1 is Initial; level 2 is Managed; level 3 is Defined; level 4 is Quantitatively Managed; level 5 is Optimizing. On the other hand, SPICE have 6 maturity levels from 0 to 6: level 0 is Incomplete; level 1 is Performed; level 2 is Managed; level 3 is Established; level 4 is Predictable; level 5 is Optimizing.

The maturity levels of the two models are actually similar in meaning although they are referred to differently and SPICE has level 0 while CMM doesn't have. In both of the models, qualitative estimation by operating the process with certain special qualities is possible up to level 3. But from level 4, the process performance data should be measured to help understand the present state, improve the process through analysis of defects and removal of the causes, and facilitate quantitative estimation and future decision-making.

2.3 Game Theory

Game theory can be defined as a study of mathematical models about conflict and cooperation between rational decision-makers. Game theory using mathematical technique was designed to analyze situations when more than 2 parties should make a decision which will have an effect on their own interest under various possibilities of cooperation or conflict. This theory was first introduced in 1944 by a physicist and mathematician, Jon von Neumann and economist Oskar Morgenstern.

A game consists of player (at least 2 players), player's strategy, outcome of decision, and payoff, numerical value of the outcome.

Outcome is dictated by who selects which strategy. Therefore, each competitor should select a strategy which can maximize his or her own interest (performance) no matter what strategy other competitor(s) will employ.

According to number of competitors, games are classified into two types: two person game (e.g. chess) and multi-person game or n-person game (e.g. poker). The most common form of games is two person zero-sum game in which the sum of all payoffs to all players is zero because one player's winning necessarily means the other's loss if there is conflicting interest between players. According to number of strategies employed by players, games are also classified into two types: finite game and infinite game (for the purpose of continuing the play). Finite two person zero sum games[8] are most widely used in theory.

This paper applies the second level meta game of non-cooperative bimatrix games, to data collecting in order to secure reliability of data to be collected and measured in developing software.

2.4 Business Psychology

Psychological factors among developers also affect developing software. For leaders of software developing organizations, it is a difficult job to induce software developers to welcome additional work of collecting metrics, other than developing software. It is pointless to attribute all of this to young generation's individualism or selfishness. Leaders of organizations should understand the group psychology in the situation[11][12][13].

For instance, when a person hears the voice of somebody with epilepsy from the next room, if he is alone, the probability for him to give help is 85%. But, when he knows another 4 people, besides him, hear the voice, the probability will be reduced to 31%. In addition, longer time will be taken to take an action. This is a very well-known experiment in psychology field[7].

By the same principle, metrics collected in small group and in large group are somewhat different in terms of reliability. That is because individuals tend to feel less sense of responsibility in larger group due to diffusion of responsibility. One of psychologies fanning this trend is bystander effect in which people think, "Someone else will take responsibility" and want to stay low profile in the group. Therefore, it is needed to secure reliability of input data by motivating developers more actively and effectively, instead of just waiting until they input accurate metrics data to be measured.

3. Collecting Measurement Data

One of the biggest differences of software and hardware development is the level of visibility provided. For example, design and implementation of software are represented as conceptual modeling and logical codes. But in the case of hardware, it is designed intuitively using 3 dimensional CAD and implemented through

combination of physical matters instead of logical codes, securing enough visibility.

Therefore, as a way to secure visibility and transparency in software development, it is strongly recommended to measure relevant data which are obtained on the course of development, and utilize the outcomes in project management.

3.1 Criteria of Data Collecting

For this paper, data were collected from 35 S/W developers of 5 organizations. Based on following 5 criteria, 27 metrics of the data were selected.

(1) Is data collecting possible?
(2) Is it contributing to improvement of Quality, Cost, Delivery, Productivity?
(3) Does it correspond to SPI (Software Process Improvement) strategy?
(4) Is the equality among developing organizations secured?
(5) Can reliability of the data be ensured?

Metrics as selected based on criteria above are shown in table 3-1(Metrics Table)

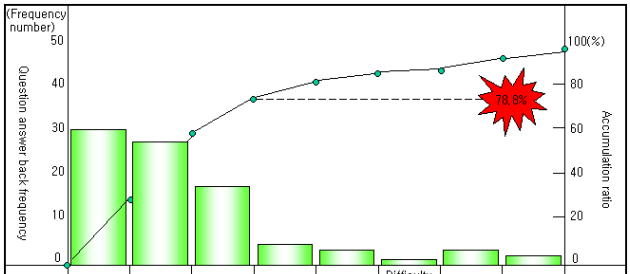
Table 3-1 Metrics Table of Measurement Data

Area	Factor	Metric	Calculation method
QUALITY	Defect	Defect density from design phase	Defect number of items of design / Number of page of the whole design document
		Defect density from module	Defect number of items of module / Total SLOC
		Defect management ratio	The management number of items which is completed / Total occurrence number of items
		Defect take out ratio	(Before release) The defect number of items which is discovered / Target discovery number of items
		Defect discovery ratio of user	(Release after 6 month ofr collections) Defect number of items / Total SLOC
	Requirement management	Requirement change ratio	Change, Add, Delete number of items / The requirement number of items which is decided
		Requirement implementation ratio	The requirement number of items where the implement is completed / The requirement number of items which is decided
		(customer)Unsatisfactory requirement number of items	The requirement number of items which is numsatisfactory / The requirement number of items which is decided(%)
		Changed number of line	Change, Add, Delete SLOC / Total SLOC(%)
	Process improvement	Design standard observance ratio	Design standard observancd number of items / Total design number of items
		Coding standard observance ratio	Sandard observance module(line) number of items / Total module(line) number of items
		Process observance ratio	Audit(checklist calculate) point
		Process improvement number of items	The improvement number of items which is reflected to a process standard
COST	Manpower cost	Manpower plan good hit ratio	The man month(MM) which becomes practice assign / The total man month(MM) which is planned
		Manpower plan good hit ratio each of phase	Each phase practice the total of the man month(MM) which is assigned / The total of the mon month(MM) which each phase is planned
	Failure cost	Defect correction time	Summation of defect correction hour(MH)
		Extra work time	Summation of over time(MH)
	Prevention cost	Teaching time	Summation of teaching time(MH)
		Review / Audit time	Until ready time summation of the hour when review/audit which is included(MH)
Delivery	Delivery observance	Delivery observance ratio	Actual accomplishment duration(month) / The accomplishment duration which is planned(month)
		Delivery observance ratio from each phase	Each phase actual accomplishment duration(month) / The accomplishment duration which is planned(month)
Productivity	Productivity	Each person productivity	KLOC / Man Month
		Each code conversion with document	Total document number of page / Total KLOC
		Each person document work	Total document number of page / Total assigned man month
	Re-use	Module re-use ratio	Re-used SLOC / Total SLOC
		Design document re-use ratio	Re-use design document number of page / Total design document number of page
	Size estimation	Size	Development the SLOC which is completed / Estimation SLOC

3.2 Obstacles to Secure Reliability of Input Data

For software developers who always feel stress of developing software, collecting metrics data is not easy. Therefore, if they are forced to collect data, data can be distorted when they input the data. Therefore, it is important to know exactly which factors are burdensome in inputting data and remove them in order to get accurate input data from developers.

Following Figure 3-1 is burdensome factors to developers obtained from a survey.



Item	Additional effort	Unsasiness	Uncertainty	Failure to meet deadline	Sense of rejection	Difficulty in understanding	Vagueness	Complicatedness	Total
Frequency occurrence number	35	32	19	6	5	3	5	4	109 (Question answer back)
Prossession ratio	32.1	29.3	17.4	5.5	4.6	2.8	4.6	3.7	100%

Figure 3-1 Pareto Chart for analysis of burdensome factors to developers

Figure 3-1 shows the results obtained from a survey of 35 software developers. The surveyed items were developed through interview with developers. Description of each item is as follows.

Additional effort: additional effort required to input measurement data, besides software development work.

Uneasiness: sense of burden that the input data may be used as a data for evaluating an individual.

Uncertainty: uncertainty about whether the quantitative management will bring actual benefits to developers.

Failure to meet deadline: Uneasiness that additional work of data input can cause failure to meet deadline.

Sense of rejection: sense of rejection toward learning about new input system.

Difficulty in understanding: lack of understanding about input items

Vagueness: Vagueness of input range

Complicatedness: complicatedness caused by data input is conducted from too many sources.

The survey shows, as shown in Figure 3-1, that three major burdensome factors when developers input data are additional effort required to input measurement data besides software development work, sense of burden that the input data may be used as a data for evaluating an individual, uncertainty about the benefits quantitative management will bring. The three factors combined account for 78.8%.

The three factors can be reduced by introducing new tools, but unfortunately, cannot be completely removed, for collecting measurement data. Therefore, this paper suggests a way to apply the second level meta game out of non-cooperative bimatrix games♣, in order to secure reliability of input data.

3.3 Analysis of Weight of Obstacles

The biggest obstacles to collecting accurate data are, as shown in Figure 3-1, boil down to three factors (additional effort, uneasiness, uncertainty). The third factor, uncertainty, is represented, here, as "profit of management" from qualitative management. After educating software developers enough about effects of quantitative management, another survey of the developers on the three factors was conducted again. As a result, weight values as shown in Table 3-2 were obtained.

Table 3-2 Weight Values for Major factors

	Additional effort	Unsasiness	profit of management
Weight Value of the data input which is reliability	-5	-3	8
Weight Value of the data input which is distorted	-1	0	-2

Table 3-3 is a matrix table of payoffs for developer A according to each strategy, based on these weight values.

Table 3-3 Distribution of Payoff according to strategy

		Strategy of developer B		
		α : Weight Value of the data input which is reliability	β : Weight Value of the data input which is distorted	δ : Data it does not input
Strategy of developer A	α : Weight Value of the data input which is reliability	0, 0	-10, 7	-5, 8
	β : Weight Value of the data input which is distorted	7, -10	-3, -3	-1, -2
	δ : Data it does not input	8, -5	-2, -1	0, 0

In the case that developer A selects α strategy, if developer B chooses α strategy too, ultimate payoff of A is 0. If B adopts β strategy, payoff for A is –10 due to false management, instead of plus gains from correct management. Finally, if B chooses δ strategy, data collecting is considered to fail because the size of the entire statistical population is too small. In this case, gains by A from quantitative management is 0, and there is no worry about being evaluated. Therefore, -5, value of additional effort for inputting data is the ultimate payoff of A. Table 3-3 shows payoffs calculated in that method.

4. Application of Game Theory

4.1 Dominant Strategy and Pure Strategy Equilibrium Point

Software developers have three strategies available as shown in Table 3-3. For developer A's side, δ strategy out of the three strategies generates the largest payoff. In this case, δ strategy of A dominates over others. Table 4-1 shows the movement diagram of developer A's payoffs in Table 3-3, based on dominant strategy.

Table 4-1 Movement Diagram of Developer A's Payoffs

		Strategy of developer B		
		α : Weight Value of the data input which is reliability	β : Weight Value of the data input which is distorted	δ : Data it does not input
Strategy of developer A	α : Weight Value of the data input which is reliability	①	②	③
	β : Weight Value of the data input which is distorted	④	⑤	⑥
	δ : Data it does not input	⑦	⑧	⑨

non-cooperative bimatrix games♣ : Two people who have a multi mind strategy do not cooperate with each other not to be, the matrix game which chooses the strategy of each one simultaneously.

.

Analysis of the movement diagram in Table 4-1 shows us that arrows converge on number ⑦, which means ⑦ is equilibrium point of pure strategy. But for developer B's side, the equilibrium point is ③. In other words, if all developers are allowed to head for equilibrium points of pure strategy with which they can maximize their own payoff, all developers will adopt δ strategy, and thus, number ⑨ will become Nash equilibrium[10]. That is why collecting measurement data itself are not easy. If there is no input data, quantitative management is impossible. As a solution to this, this paper adopts a somewhat forceful way for measurement data entry.

The way involves inducing developers to input measurement data by imposing penalty on those who fail to input data. The movement diagram of this case corresponds to movement diagram of payoff when the third strategy, δ , of developer is excluded from Table 4-1. The movement diagram is shown in Table 4-2.

Table 4-2 Movement Diagram of Payoff when Strategy δ is Excluded

		Strategy of developer B	
		α : Weight Value of the data input which is reliability	β : Weight Value of the data input which is distorted
Strategy of developer A	α : Weight Value of the data input which is reliability	①	②
	β : Weight Value of the data input which is distorted	④	⑤

In Table 4-2, equilibrium point of pure strategy for developer A's payoff is number ④. Like case in Table 4-1, all of developers can maximize their own payoff when they select strategy β . Therefore, if all of them pursue for strategy β , number of cases ⑤ will become Nash equilibrium point and the ultimate payoff of developers will be ⑤. But, number ⑤ is non-Pareto optimal outcome as ① generates more payoff than ⑤. In other words, in Table 4-2, non-Pareto optimal outcome ⑤ is selected instead of Pareto optimal outcome ①. This result shows that individual rationality based on principle of domination and group rationality based on Pareto principle do not correspond.

4.2 Prisoner's Dilemma

Like the case in Table 4-2, non-zero sum game which has an equilibrium point, but non-Pareto optimal is called Prisoner's Dilemma[9]. It is referred as to the case in which players come to select non-Pareto optimal strategy if all players pursue their own maximum payoff, even though there is a Pareto optimal strategy available which provides maximum payoff to every one [5].

4.2.1 Exclusion of Strategy β

As a solution to this dilemma, strategy β can be excluded from Table 4-2, like from Table 4-1 in order to obtain Pareto optimal strategy. But exclusion of strategy β is impossible. Exclusion means here preventing developers from entering false data. This can be accomplished by examining reliability of all the input data and imposing penalty on developers who input false data. But it is difficult for developers to thrust away the temptation of strategy β because they know the fact that examining every input data is, in effect, impossible. Therefore, excluding strategy β is not a proper way.

4.2.2 Meta Game

In the case of the first level meta game where developer B decides sub-strategy according to strategy choice of developer A, A has 2 strategies(α , β) and B has following 4 strategies.

α α : To input reliable data regardless of developer A's strategy.

α β : To select the same strategy as expected A's strategy.

β α : To select the opposite strategy to expected A's strategy.

β β : To select false data regardless of A's strategy.

Table 4-3 is the matrix representing this situation.

Table 4-3 First level Meta Game

		Strategy of developer B			
		αα	αβ	βα	ββ
Strategy of developer A	α	0, 0	0, 0	-10, 7	-10, 7
	β	7, -10	-3 -3	7, -10	-3 -3

In the matrix above, strategy β of developer A does not dominate over strategy α anymore and strategy β β of developer B dominates over the other three ones. Thus, strategy β of A, strategy β β of B are the only equilibrium point. But this does not necessarily mean cooperation (input of reliable data) between the two developers. In other words, solution for the first level meta game is not cooperative strategy.

As Pareto optimal outcome is not obtained, developer A and B conduct the second level meta game in which developer B takes a sub-strategy and developer A selects a sub-strategy according to B' s sub-strategy. In this case, developer A has 16 strategies available as shown in Table 4-4.

Table 4-4 Second level Meta Game

		DEVELOPER B			
		αα	αβ	βα	ββ
DEVELOPER A	αααα	0, 0	0, 0	-10, 7	-10, 7
	ααaβ	0, 0	0, 0	-10, 7	-3 -3
	ααβα	0, 0	0, 0	7, -10	-10, 7
	ααββ	0, 0	0, 0	7, -10	-3 -3
	αβαα	0, 0	-3 -3	-10, 7	-10, 7
	αβαβ	0, 0	-3 -3	-10, 7	-3 -3
	αββα	0, 0	-3 -3	7, -10	-10, 7
	αβββ	0, 0	-3 -3	7, -10	-3 -3
	βααα	7, -10	0, 0	-10, 7	-10, 7
	βααβ	7, -10	0, 0	-10, 7	-3 -3
	βαβα	7, -10	0, 0	7, -10	-10, 7
	βαββ	7, -10	0, 0	7, -10	-3 -3
	ββαα	7, -10	-3 -3	-10, 7	-10, 7
	ββαβ	7, -10	-3 -3	-10, 7	-3 -3
	βββα	7, -10	-3 -3	7, -10	-10, 7
	ββββ	7, -10	-3 -3	7, -10	-3 -3

In Table 4-4, there is no dominant strategy for developer B while strategy β α β β of developer A dominates over others. Therefore, developer B realizes that A will select β α β β , and thus selects strategy α β favorable to B when A employs β α β β . Accordingly, β α β β of A, α β of B becomes equilibrium point of the second level meta game, as shown in Table 4-4. So to speak, the best way for A is to believe that developer B will select the same strategy as him or her and take a cooperative strategy, and developer B knows the fact and selects the same strategy as A. Consequently, the players of the game have no choice but choosing a cooperative strategy in order to get maximum payoff. In this way, Pareto optimal solution for the second level meta game is obtained.

5. Simulation Study

Robert Axelrod maintained a natural evolution from competition to cooperation in his book 『The Evolution of Cooperation』 [6]. The name of the game here is to demonstrate the question, "how can cooperation be developed in the world of fundamentally selfish agents?" This question involves following three questions: How can cooperation start? ; How can cooperative strategies survive better than non-cooperative ones? ; Which cooperative strategy can produce the best payoff and how can this dominate over the others [6]. He demonstrated these questions by applying prisoner's dilemma game.

This paper also applies the theory of Robert Axelrod to software development in order to change the developers behavior from competition to cooperation, and simulates this, using 『winpri』 , a software developed by Philippe Mathieu and Fredderic Grignion.

Following figures are the outcomes obtained by simulating strategies selected by software developers through 『winpri』 after explaining 15 strategies of Robert Axelrod to them.

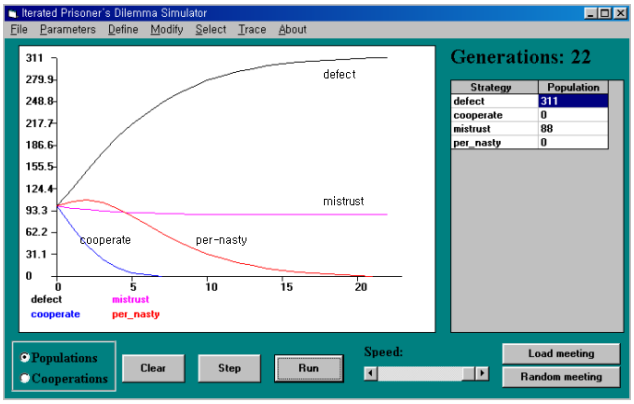


Figure 5-1 Cooperative Strategy among Defection Strategies

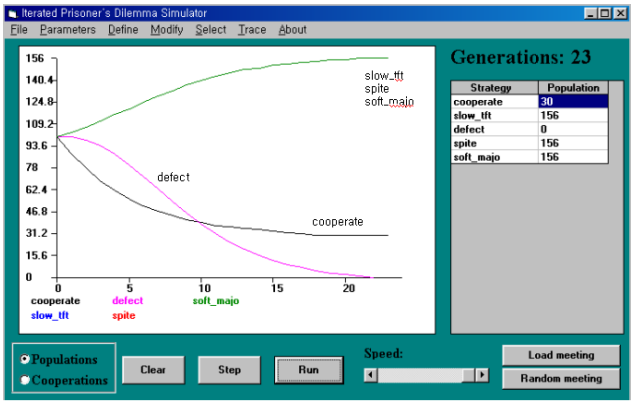


Figure 5-2 Defection Strategy among Cooperative Strategies

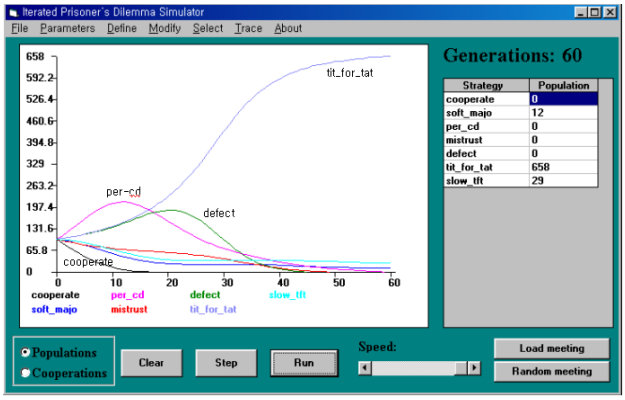


Figure 5-3 Dominant Strategy among Strategies

Figure 5-1 shows a cooperative strategy among defect strategies, and Figure 5-2 shows the payoff of a defection strategy among cooperative strategies. Finally, Figure 5-3 shows situations when cooperative strategies are mixed with defection strategies. In the early part of the game, defection strategies, "per-cd" and "defect" are seemed to dominate. But ultimately, "tit_for_tat" strategy (cooperating but retaliating partner's defection with defection) produces the biggest payoff.

The simulation shows that cooperative strategy is dominant over selfish (defection) strategy. It also shows that reciprocal cooperation to respond the other's defection with defection is more efficient than unconditional cooperation. As a result of educating these results to developers enough, and then collecting data again, it was found that the reliability of data is improved by 9.8%. This paper was analyzed through point estimation and interval estimation of statistical inference, for verification of its reliability.

6. Conclusion & Feature work

In software development, quantitative process management is one of the most common techniques to improve visibility (transparency). CMM and SPICE also recommend that quantitative process management be adopted when the maturity of the software process reaches a certain level. But if reliability of data is damaged, quantitative process management becomes pointless because every activity for quantitative process management is based wholly on metrics data. Therefore, it is necessary to verify reliability of data. Many ways to analyze the reliability have been studied. Most of them, however, are just analysis methods for already collected data.

Therefore, if the analysis finds that the existing data are not reliable, the data should be abandoned and new data be collected again, causing need to consume much time and effort. Therefore, as a way to improve reliability of measurement data to be collected, this paper deals with a technique to induce developers to input accurate data in the first place, instead of analyzing reliability of already collected data.

This paper analyzes psychological factors affecting data input (e.g. psychology that somebody else will take responsibility), additional efforts needed to input data, besides software development, and the concern that the data may be used to evaluate individuals, and studies on the solutions.

This paper finds the solution in the second level meta game of cooperative bimatrix games, applies the cooperation evolution theory of Robert Axelrod to the game, and demonstrates the solution to software developers by simulating the results of the application through 『winpri』 developed by Philippe Mathieu and Fredderic Grignion. As a result, reliability of measurement data is improved by 9.8%, compared to data collected with traditional methods.

The next study may well deal with organizational behavior theory defining the relation of project manager, software developer, and SPI for successful project management through application of game theory.

Reference

[1] Sang-Pok Ko, "A Study on the Measurement for Embedded Software", SERP. Vol. 2, pp. 635-638 June 2003.

[2] Paulk. M. C. et al. *The Capability Maturity Model : Guidelines for Improving the Software Process*. Addison-Wesley Pub Co., 1995.

[3] ISO/IEC 15504 TR2. *Software Process Assessment and Capability determination*. ISO/IEC 1998.

[4] Roser S. Pressman. *Software Engineering A Practitioner's Approach*. McGraw-Hill Pub Co., Fourth Edition 1998

[5] R.Axelrod. *The Evolution of Strategies in the Iterated Prisoner's Dilemma*. Pitman. London. 1987

[6] R.Axelrod. *The Evolution of Cooperation*. Basic Books. New York. 1984

[7] Ji-Hyun Ha(A psychiatrist). *The business psychology which is piquant.* Cheang-Lim Pub Co., Seoul. 2004

[8] Raghavan, T., "Zero-sum two-person games", *Handbook of Game Theory*, Volume 2, pp. 736-759, 1994.

[9] Rapoport, A., and A. Chammah, Prisoner's Dilemma, University of Michigan

[10] Nash, J., "Equilibrium points in n-person games", *Proceedings of the National Academy of Sciences USA*, 36, pp. 48-49, 1950

[11] Lucas, W., Game theory and Its Applications, American Mathematical Society, 1981.

[12] Milnor, J., "Games against nature", in Game Theory and Related Approaches to Social Behavior, Wiley, 1964.

[13] Rapoport, A., Mathematical Models in the Socialand Behavioral Sciences, Wiley-Interscience, 1983.

Development of E-tutorial for Open Source UML Tool (ArgoUML) Using Drupal as a CMS

A. Avishek Saha[1], **and B. Amee Kiran Lakhani**[2]
[1]Msc-CA,Symbiosis International University, Pune, Maharashtra, India
[2]Msc-CA,Symbiosis International University, Pune, Maharashtra, India

Abstract *– E-tutorial, the new-age learning technique, quite customized and convenient in nature and has become a booming industry worldwide giving an impetus to both career and study. Open source provides a development methodology; for free software, the users' essential freedoms: the freedom to run it, to study and change it, and to redistribute copies. This E-tutorial effort is for those learners who want to construct high quality software applications using Unified Modelling Language (UML), commonly used to visualize and construct systems which are software intensive. This paper focuses on the designing and development of E-tutorial on a leading open source UML modelling tool (ArgoUML) using Drupal as Content Management System. An illustration of a case-study is also provided which will help in understanding the tools and Object Oriented Analysis and Design concepts. It can also be adapted for classroom learning wherein the students and teacher can work simultaneously on a UML tool, its diagram or a case study.*

Keywords: E-tutorial, Open Source, ArgoUML, Content Management System, Drupal, MySQL

1 Introduction

Today, there are diverse proprietary software development tools available for building high quality software using Object Oriented concepts. The challenges for the new learner are accessibility, training, experience, affordability, ease of use etc. This E-tutorial illustrates the concepts of Object Orientation, Unified modelling language techniques, and a complete tutor of ArgoUML with the help of solved case study. This E-tutorial evolves using UML modelling tools, use of Drupal as a Content management system and essentials of Object Oriented Analysis and Design concepts. E-tutorial is the blend of Open Source technologies to enable learner to learn anytime and anywhere, just-in-time information and guidance. In addition, this website is an excellent solution for every learner, who wants to adapt to Object Oriented Analysis and Design concepts using Open Source UML modelling tool (ArgoUML).

2 Methodologies, Technologies and Tools

For developing the unique E-tutorial on ArgoUML, the Open Source Content Management System as Drupal, and administration is managed by MySQL. This is an intellectual piece at least cost and the end user can use this E-tutorial through the web-site freely. That is completely an OPEN SOURCE approach.

2.1 OOAD Concepts

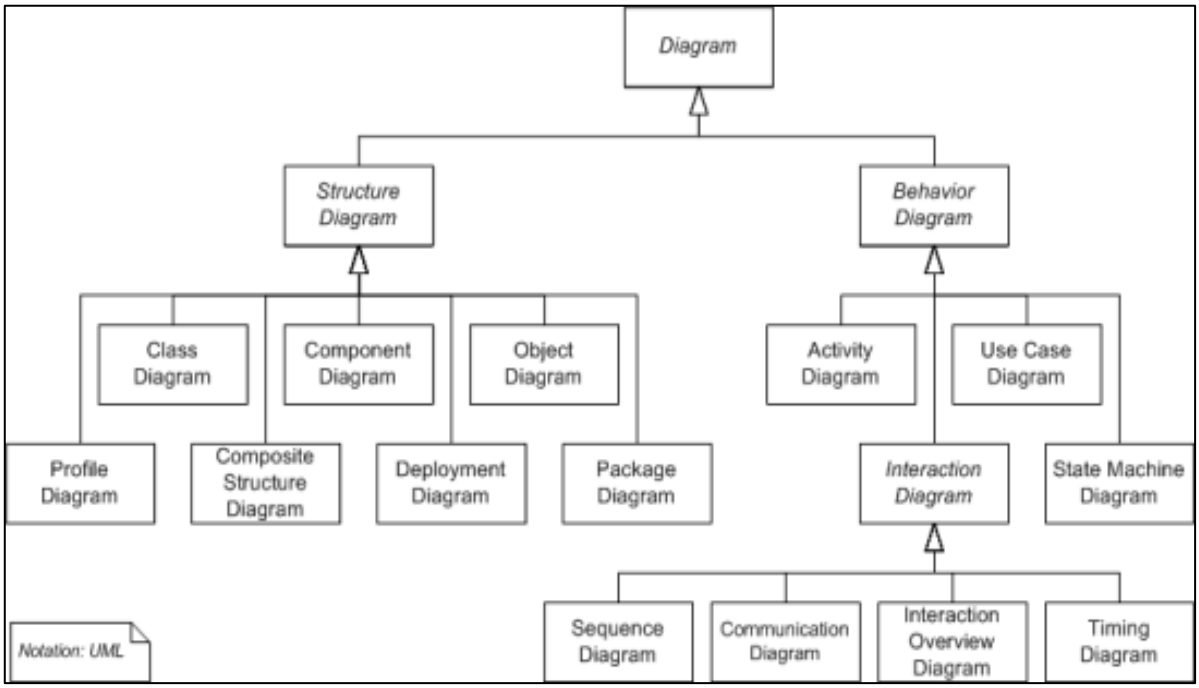
OOAD Concepts are the essence of object oriented analysis and design to emphasis the problem domain and provide the logical solution from the perspective of objects (thing, concepts or entities).

2.2 Unified Modelling Language (UML)

UML allows the modelling of business processes. A business process is a set of logically related tasks executed to achieve a business result. To achieve the best business processes developed under OOAD concepts, UML play important role in developing object oriented software and the software development process for business processes. It is a graphical notation to express the design of software projects. UML diagrams represent two different viewsof a system model. Static (or structural) view which emphasizes the static structure of the system using objects, attributes, operations and relationships and Dynamic (or behavioral) view which emphasizes the dynamic behaviour of the system by showing collaborations among objects and changes to the internal states of objects.

2.3 ArgoUML

ArgoUML is the leading open source UML modelling tool and includes support for all standard UML diagrams. This diagramming application is written in Java and released under the open source BSD License. By virtue of being a Java application, it is available



on any platform supported by Java. It is able to create and save all standard UML diagrams. This has the ability to reverse engineer compiling Java code and generate UML diagrams for it.

Jason Robbins initially built ArgoUML as a test bed for ideas in cognitive psychology and their applications to software design. These features are unique to ArgoUML and include design critics, corrective automations, to-do lists, usage-based tool adaptation and design.

2.4 Content Management System

A content managemnt system (CMS) is a collection of procedures used to manage work flow in a collaborative environment. These procedures can be manual or computer-based. The procedures are designed to:

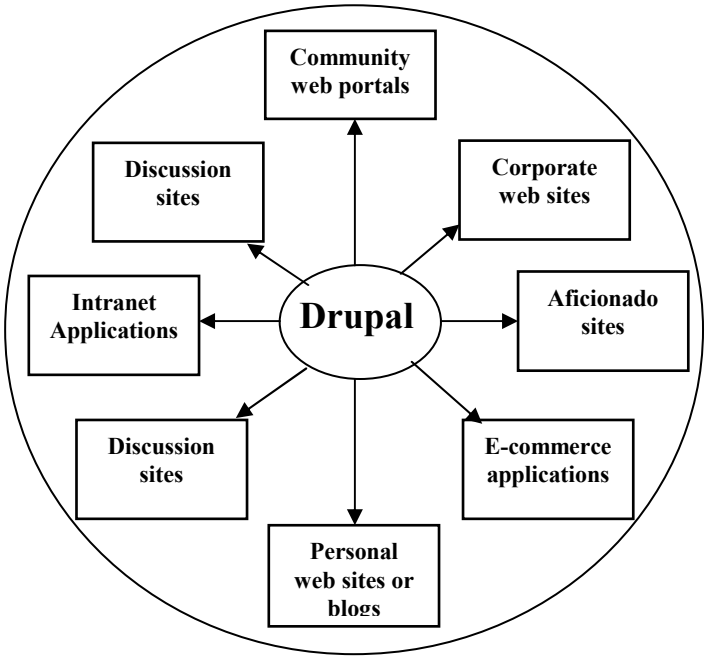
- Allow for a large number of people to contribute to and share stored data
- Control access to data, based on user roles. User roles define what information each user can view or edit.
- Aid in easy storage and retrieval of data
- Reduce repetitive duplicate input
- Improve the ease of report writing
- Improve communication between users

2.5 Drupal

It is a content management system using which this E-tutorial was developed. There are important built-in functionality which can be combined with thousands of freely available add-on modules.

Features which are of importance are as follows:

- Electronic commerce
- Blogs
- Collaborative authoring environments
- Forums
- Peer-to-peer networking
- Newsletters



- Podcasting
- Picture galleries
- File uploads and downloads

Before the use of Drupal, the survey of all open source Content Management System and the web servers available were surveyed, The result of the survey was based on parameters such as Built-in Applications, Security Management, Performance, Commerce, Ease of Use, and Support, rated highest in its class for 2010. Following procedure is presented out of experience while working with Drupal as CMS, which has installation of Drupal, Setting Up the MySQL Database for administration, installing the module which are required from vast assortment of modules of Drupal, Installing new themes for good GUI, generating content on ArgoUML for the E-tutorial.

2.6 Installing Drupal

Download Drupal from http://drupal.org. Download Drupal 6.15, which is the current working module and install it on our machines.

Basic requirements of Drupal are: Resources, Web server, Database server, PHP. A variety of packages enable WAMP (Windows, Apache, MySQL, PHP/Perl/Python) to be downloaded (and, if not portable, installed) on a Windows-based computer. And hence, one of the pre- requirements to install Drupal on window's is the installation of WAMP server on windows.

1. Link to Download WAMP is www.wampserver.com/en/download.php
2. To successfully install WAMP, click the link provided for additional versions and download the WampServer2.0h as the WampServer2.0i version that you are prompted to download from the WAMP website is *not compatible* with Drupal.
3. Once installed, your WAMP server will prompt you to use the default setting LocalHost and then have a second data entry box asking for an e-mail address with a generic **you@yoursite.com** placeholder. Leave the LocalHost setting and enter your e-mail address to the data box.
4. Left Click on the WAMP Server Icon in the System Tray.
5. Click on the PHP Menu Selector.
6. Select the httpd.conf icon.
7. As mentioned earlier, Drupal installation files are non .exe files. The Drupal install files come in a compressed .tar or .gzip format.
8. Once downloaded and uncompressed, copy the Drupal file folder into your WAMP Folder. i.e. C:/WAMP/www/drupal
9. Left-click on the WAMP Icon in the System Tray. Then select the Apache menu, and the first file shown should be your httpd.conf file.
10. Once opened locate the line that reads.

```
ServerName LocalHost:80
```

Now copy and paste the following code below the ServerName line you just found.

```
NameVirtualHost *:80
<VirtualHost *:80>
DocumentRoot "c:/wamp/www/"
ServerName localhost
</VirtualHost>
<VirtualHost*:80>
DocumentRoot
"c:/wamp/www/mysite"
ServerName mysite.localhost
ServerAlias mysite
</VirtualHost>
```

2.7 Setting Up the mySQL Database

- The use of phpMyAdmin to set up our SQL account. To do this open your browser and type in http://localhost/phpmyadmin
- Once loaded, type the name for your database into the Create New Database field.
- Lastly add a custom user, click the privileges tab, set your username and password and close your browser.
- After successful configuration of WAMP we need to configure Drupal
- Once Drupal is configured we need to install new modules

2.8 Installing new themes

The first step to installing the module is to find the required module at drupal.org through their vast assortment of modules. Following modules were installed in the project. Each module used to create this E-tutorial along with its brief description is given below:

- *admin_menu:* Administration menu module providesa theme-independent administration interface
- *cck:* (Content Construction Kit) this allows you to add custom fields to nodes using a web browser.
- *emfield:* This extensible module will create fields for node content types that can be used to display video, image, and audio files from various third party providers. When entering the content, the user will simply paste the URL or embed code from the third party, and the module will automatically determine which content provider is being used. When displaying the content, the proper embedding format will be used.
- *filefield:* FileField provides a universal file upload field for CCK. It is a robust alternative to core's Upload module and an absolute must for users uploading a large number of files. Great for managing video and audio files for podcasts on your own site.
- *wysiwyg:* Wysiwyg module allows you to use

client-side editors to edit content in Drupal. It simplifies installation and integration of editors and allows assigning an editor to each input format. Wysiwyg module replaces all other editor integration modules and no other Drupal module is required.

- *views:* provides a flexible method for Drupal site designers to control how lists and tables of content are presented.
- *pathauto:* The Pathauto module automatically generates path aliases for various kinds of content (nodes, categories, users) without requiring the user to manually specify the path alias. This allows you to get aliases like /category/my-node-title.html instead of /node/123. The aliases are based upon a "pattern" system which the administrator can control.
- *token:* Tokens are small bits of text that can be placed into larger documents via simple placeholders, like %site-name or [user]. The Token module provides a central API for modules to use these tokens, and expose their own token values.
- *imagefield:* ImageField provides an image upload field for CCK. ImageField is a CCK-based alternative to the legacy Image project. It boasts features such as multiple images per node, resolution restrictions, default images, and extensive Views support

2.9 Installing new themes

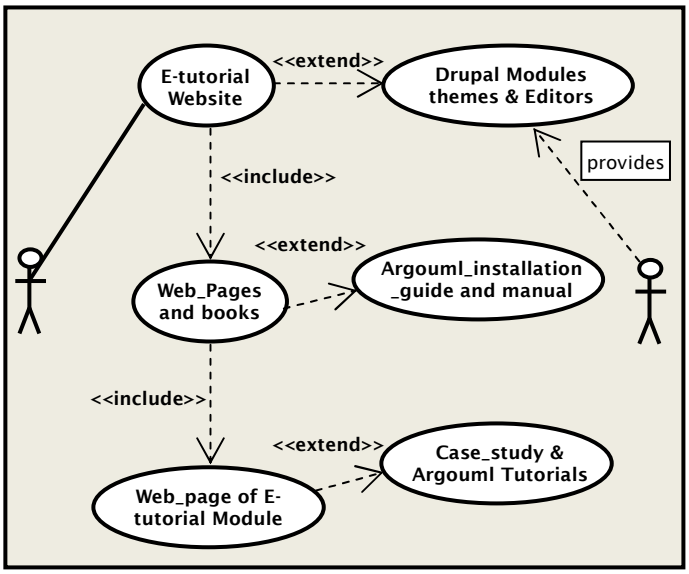
Once we installed Drupal and started to come into terms with it we wanted to customize the way it looks. A good GUI is very essential for all websites. We had to select the best GUI, i.e. suitable for our E-tutorial and which would attract more and more users. There are several themes which can be downloaded from the Drupal web site. We checked out few themes and selected one theme which we found was suitable for our website. and we proceeded towards its installation.

2.10 Building content of the website

Once the work on the GUI and structure of website was ready, we started working on the content generation part. Before developing the content, it was necessary to learn and adapt OOAD concepts. The research work was on implementation of OOAD concepts using ArgoUML and created various UML use case diagrams. The next step was building the content for the E-tutorial which explain in brief the steps to draw each diagram and explaining the different tools and functionality provided by ArgoUML. Further work was based on creating a manual for ArgoUML, where delivering a solved case study to give a broader view of the usage of ArgoUML.

3 Design of e-tutorial

The designed website named ArgoUML E-tutorial as represented in the diagram



3.1 E-tutorial website

This website aims to provide the tutorial on ArgoUML. It is expected that this Web enabled tool can provide a very cost effective solution, at affordable level to educational institutes.

3.2 Drupal modules and themes

This website is created using DRUPAL as Content management software. In order to create the website different Drupal modules and a theme (i.e. Acquia Marina) was used as per requirement.

3.3 Webpage and books

After installing necessary modules each webpage was created in a book structure. Each webpage were added as a child page and hence ArgoUML tutorial was created.

3.4 ArgoUML installation guide & manual

This website provides a detailed manual for installation of ArgoUML.

3.5 Webpage of e-tutorial module

Each child page of this website provides a tutorial

for ArgoUML. It was created using various modules.

3.6 Case study and ArgoUML tutorial

A Guide to use each and every tool provided by ArgoUML for UML modelling and its description is given. This website also provides Solved Case-study and step by step instructions to draw various UML diagrams.

4 Issues, challenges and mitigation actions

4.1 Technological

This section includes various technological issues and constraints faced and overcome while working on Drupal for the developing this E- tutorial website.

4.1.1 Installing Drupal

There is no .exe file for installation of Drupal. He we had to research and study how to install Drupal

- *Non-availability of manual :* No proper manual on installation of Drupal is available on drupal.org.
- *Creating a database in Drupal :* After in depth study and research about Drupal and its installation we had to configure the Drupal website creating database for the site.

  Before running the Drupal installation script, create a database on MySQL server. This can be accomplished using any MySQL database management tool, whether a graphical program such as phpMyAdmin or MySQL's built-in monitor program, as we will do here, at a command line:

  1. Mysql --password --user=root
  2. After entering the MySQL server's password, we see a welcome banner and a command prompt:
  3. Welcome to the MySQL monitor. Commands end with; or \g.
     Your MySQL connection id is 4
     Server version: 5.0.51a-community-nt
     MySQL Community Edition (GPL)
     Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
     mysql >
  4. At the prompt, enter the commands to create the database (in this example, imaginatively named "drupal_db") and a userofthe database ("drupal_user"), with a password and all privileges within the database:

     CREATE DATABASE drupal_db; GRANT ALL ON drupal_db.*
     TO drupal_user@localhost

     IDENTIFIED BY 'password';

4.1.2 Database configuration

- Enter whatever values earlier chosen as database name, username, and password.
- If all goes well, the installation of the Drupal files should progress until it is completed.
- When the file installation is finished it will be ready to set the initial site configuration

4.2 Selection of Modules

Studying each and every module from 20000+ modules. The knowledge of functionality of each and every module was necessary so as to obtain desired objective for creating this website. Studying the working of each module testing them by installing and using each of them and then using the best out of them, were a few task performed.

For eg. For inserting an image in drupal, the following modules are available: image gallery drupal, ImageCache, Image, ImageField, ImageAPI, Upload Image, Image Slider, etc. Each module functions differently. Like image gallery allows user to view images in slide show, image module doesn't allow the user to edit, resize or crop the image. Hence to made use of image cache as none other module was satisfying the needs. But before coming to this conclusion all other modules were tried & tested.

Inter-dependency of various modules: There are many inter-linking modules in Drupal. Many modules, image cache, need to have few other modules installed in prior, to get the desired module installed successfully. Moreover, these pre-requisite modules may need some other modules to get chain. So it was a tedious task to install and understand the inter-linking modules.

- *Installing the modules in Drupal:* The modules when extracted would sometimes behave unusually and not work, in which case they had to be extracted again.

- *Inserting an Image:* For image insertion, an image cache needs to be made using the "image cache" module. This helps in putting all the images that has to be used in Drupal site pages together.

- *Using CONTENT MANAGEMENT SYSTEM:* For managing the content, its structure was studied. In this project, the use of the "book" module to create a website of several pages

4.3 Argo UML

- There is no manual available which gives detailed description of each and every tool used in ArgoUML.
- There are no step by step instructions to draw the UML diagrams.
- There is no solved case-study available using ArgoUML.

5 E-tutorial beneficiary

5.1 Learners

It involves some form of interactivity, which may include online interaction between the learner and their teacher or peers. This website is created and designed keeping in mind various users and it is best suitable for beginner's i.e. students and other users who wish to learn using ArgoUML.

5.2 Teachers

OOAD concepts are taught in most of Computer studies colleges and universities. And most of them make use of ArgoUML as a UML modelling tool. Hence various institutes and universities can make use of this website to teach ArgoUML. A large number of students can also be targeted for teaching through this website with ease.

5.3 Industry

This E-tutorial can also be used by industry if they want to make use of ArgoUML in their project. It will be easier for all Project manager's and team members to get the quick shot at any time of the life cycl

6 Conclusion

A Web enabled E-tutorial on ArgoUML was developed successfully. With Industry, Teachers of OOAD and Students who want to implement the OOAD concepts will be benefited. The industry highly appreciated CMS, played an import role in delivery of this E-tutorial.
The various modules and themes of Drupal were tested and adopted as per the requirements of the E-tutorial. To make the website user friendly, a case study and the steps of each diagrams were explained. It is completely a Open Source solution for teaching and learning of UML with the help of Content Management System as Drupal.

7 References

[1] Russ Miles, Learning UML 2.0
[2] Grady Booch, OBJECT-ORIENTED ANALYSIS AND DESIGN
[3] Jacob Redding , Beginning Drupal
[4] Dan Pilone, UML 2.0 in a Nutshell
[5] Michael Alexander, CONTENT MANAGEMENT SYSTEM Made Simple Guide for Beginners
[6] http://www.dundee.ac.uk/learning/dol/ELS_final_report.pdf (student survey)
[7] http://argouml.tigris.org/downloadcount.html (argoUML survey)
[8] http://Content Management System-software-review.toptenreviews.com/ (Survey for the best CONTENT MANAGEMENT SYSTEM 2010)
[9] http://argouml.tigris.org (argoUML website)
[10] http://drupal.org (drupal website) [11] http://en.wikipedia.org/
[12] http://php.net
[13] http://www.siteground.com
[14] www.developer.com
[15] www.sharpprogrammer.com
[16] Jon Holt Institution of Electrical Engineers (2004). UML for Systems Engineering: Watching the Wheels IET, 2004 ISBN 0863413544. p.58

SESSION

SOFTWARE TESTING

Chair(s)

TBA

Generating Verifiable Test Scenarios

J.-P. Corriveau[1] **and W. Shi**[2]
[1]Computer Science, Carleton University, Ottawa, Ontario, Canada
[2]Business and IT, UOIT, Oshawa, Ontario, Canada

Abstract - *Many approaches that address scenario testing do so using models semantically distant from an implementation under test (IUT). While test paths can be generated from such models, these paths typically do not include path sensitization data and are not testable against an actual execution of an IUT. In this paper, we explain how the Validation Framework (VF) we have developed models scenarios and parse these into test scenarios. The latter do include path sensitization data and are verified by the VF against an actual execution of an IUT.*

Keywords: validation, model-based testing, scenarios

1 Introduction

A quality-driven approach to software development and testing demands that, ultimately, the requirements of stakeholders be *validated* against the *actual behavior* of an implementation under test (IUT). That is, there needs to be a systematic (ideally objective and automated) approach to the validation of the requirements of the stakeholder against the *actual behavior* of an IUT [1, 2]. Unfortunately, most often, there is no such systematic approach to validation. Quite on the contrary, in practice, testers mostly carry out only extensive *unit testing* [3]. In this paper, we are instead concerned with scenario testing.

Model-Based Testing (MBT) involves the derivation of test cases from a model (or set of models) that describes some of the aspects of an IUT. More precisely, an MBT method/tool uses various algorithms and strategies to generate tests from a behavioral model of an IUT. Scenarios [3, 4, 5, 6, 7] constitute one type of behavioral model (another being state machines [3]). Such a model is usually a partial representation of the IUT's behavior, 'partial' because the model abstracts away some of the implementation details. Tests cases derived from such a model are functional ones expressed at the same level of abstraction as the model. Such test cases are grouped together to form an abstract test suite. Most importantly, such an abstract test suite *cannot* be directly validated against the execution of an IUT because the test cases are not at the same level of abstraction as the code. Indeed, several MBT tools claim to offer test generation and test execution. But it is important to understand that in the context of MBT, 'test execution' generally consists in *symbolic execution*, that is, is carried out using a (typically state-based) model of the IUT, *not* the actual IUT. For example, Spec Explorer [8, 9, 10] stands out as an industrial-strength MBT tool with a specification language (Spec #) rich enough to capture a detailed state-based model of an IUT's behavior, as well as user-defined scenarios. But, the latter are not taken as defining a grammar of valid and invalid sequences of procedures of an IUT. Instead, these scenarios are interpreted in terms of states of the model and are validated via their symbolic execution. In contrast, here we propose an approach to scenario testing that generates test scenarios that can be validated against the execution of an IUT. To do so, we first briefly summarize in the next section the Validation Framework (VF) we have introduced elsewhere [2, 11]. The key characteristic of that tool is that implementation-independent specifications are bound to actual types and procedures of an IUT in order to enable the validation of the specification model against the execution of an IUT. Examples of scenarios in the VF are presented in section 3 and their testing is discussed in section 4.

Before concluding this introduction we remark that tools meant to only capture models (e.g., IBM's Rational Rose[12]) are not relevant to this paper, as they are not meant to address scenario testing. The same observation holds for *test automation frameworks* (e.g., IBM's Rational Robot [13]) in which there is no test generation from models (for the simple reason that such frameworks are not model-based). In the same vein, code-based testing tools (such JAVA's JUnit [14] and AutoTest [15]) mostly allow for *unit* tests (i.e., tests pertaining to a procedure, as opposed to tests addressing scenarios) to be specified in, or automatically generated from, code. Unit tests are semantically much simpler than scenario tests. Furthermore, such unit tests are implementation-specific and difficult (nay impossible) to trace back to the requirements of stakeholders (which are typically implementation-independent). For these reasons, code-based testing tools will not be discussed further here.

2 VF: An Alternative for MBT

In order to validate the requirements of a stakeholder against the actual behavior of an IUT, it is necessary to have a specification language from which tests can be generated and executed 'against' an actual IUT (as opposed to a model of the latter). We have described such an approach and its corresponding tool, the VF, at length elsewhere [2, 11, 16]. Our VF operates on three input elements. The first element is the Testable Requirements Model (hereafter TRM). This model is expressed in ACL, a high-level general-purpose requirements contract language. We use here the word 'contract' because a TRM is formed of a set of contracts, as

will be illustrated in the next section. ACL is closely tied to requirements by defining constructs for the representation of scenarios, and design-by-contract constructs [17] such as pre and post-conditions, and invariants. The second input element is the candidate IUT against which the TRM will be executed. This IUT is a .NET executable (for which we do not require the source code). *Bindings* represent the third and final input element required by the VF. Before a TRM can be executed, the types, responsibilities, and *observability* requirements of the TRM (see next section) must be bound to concrete implementation artifacts located within the IUT. A structural representation of the IUT is first obtained automatically. Our binding tool, which is part of the VF, uses this structural representation to map elements from the TRM to types and procedures defined within the candidate IUT. In particular, our binding tool is able to automatically infer most of the bindings required between a TRM and an IUT [11, 16]. Such bindings are crucial for three reasons. First, they allow the TRM to be independent of implementation details, as specific type and procedure names used with the candidate IUT *do not* have to exist within the TRM. Second, because each IUT has its own bindings to a TRM, several candidate IUTs can be tested against a single TRM. Finally, bindings provide explicit traceability between a TRM and IUT.

Once the TRM has been specified and bound to a candidate IUT, the TRM is compiled. Upon a successful compilation, all elements of the TRM have been bound to IUT artifacts. The result of such a compilation is a single file that contains all information required to execute the TRM against a candidate IUT. (Details lie beyond the scope of this paper and are available at [11].) The validation of a TRM begins with a structural analysis of the candidate IUT, and with the execution of any *static checks* (e.g., a type inherits from another). Following execution of the static checks, the VF starts and monitors the execution of the IUT. The VF is able to track and record the execution paths generated by the IUT, as well as execute any dynamic checks, and gather user-specified metrics [11] indicated by the TRM. The execution paths are used to determine if each scenario execution *matches* the grammar of responsibilities corresponding to it within the TRM (see next example). Next, metric evaluators are used to analyze and interpret any metric data that was gathered during execution of the IUT. All of the results generated from execution of the TRM against the candidate IUT are written to a Contract Evaluation Report (CER). The generation of the CER completes the process of executing a TRM against a candidate IUT. The CER indicates where the candidate IUT matches the TRM, and where any deviations from the TRM were observed. For example, when a pre- or post-condition fails, the execution proceeds but that failure is logged in the CER. Also, when a scenario is executed by an IUT, the specified grammar of responsibilities must hold in order for the scenario to be considered to have succeeded. That is, for success, the responsibilities that compose the scenario must be executed in an order that satisfies the grammar. If the scenario cannot be executed, or responsibilities/events that are not defined by the scenario are executed, then the scenario is deemed to have failed. This mismatch is also reported in the CER.

The key point of this overview is that once a TRM is bound to an IUT, all checks are automatically instrumented in the IUT whose execution is also controlled by the VF. As explained above, this enables verifying that actual sequences of procedures occurring during an execution of an IUT 'obey' the grammar of valid sequences defined in ACL scenarios. As we will illustrate in section IV, scenario testing proper goes beyond this sort of validation.

3 An Example

In order to discuss scenario testing, we must first illustrate the semantics of the language we use to specify a TRM, especially those features that pertain to scenario testing. Consequently, we now present excerpts of a medium-size case study that deals with how students register in their courses at a university and how they obtain grades for these courses. Our intent is not to motivate the contracts used, nor to explain at length how this example is processed. Instead, we aim at providing the reader with a substantial (i.e., non-trivial) example in order to a) illustrate the semantics of ACL, especially pertaining to scenarios and b) subsequently discuss the basics of scenario testing in the VF. (The complete example is available at [11] and is 49 pages long. Most importantly it does compile and run, as is the case with all other examples in [11]. The reader may verify this claim by downloading the tool and trying it out!) This example is at a level of abstraction similar to models specified in Spec# [8, 9], and appears to most readers to be surprisingly low (in fact akin to programming). We remark that the level of abstraction of specification languages for MBT varies considerably [10]. In particular, we must emphasize that Spec# [9] (which supports Spec Explorer [8], the only industrial-strength MBT tool we know) works at a low level of abstraction (similar to that of ACL) because it deals with real systems, not toy examples. For such systems, semantic expressiveness and scalability are essential.

In the following example, we use the // and /* */ to provide comments in context, as this facilitates the presentation of ACL's features. Our example starts with the Course contract, which represents a single university course. A course is created by the university, and consists of a name, code, a list of prerequisites, a list of students currently enrolled in the course, and a size limit on the number of students that can take the course.

```
Namespace Examples.School{
Contract Course{
/* Once the contract Course is bound to a type of the IUT, each time an instance of this type is created, a new instance of contract Course is associated with it. A contract can be bound to several types.
Parameters can be left unspecified until run-time, using the keyword InstanceBind, in which case, each time an instance
```

of a class bound to this contract is created, the VF will prompt the user for all required parameter values. */

```
Parameters
{ Scalar Boolean InstanceBind HasFinal =
   { default true, false };
// other parameters have been omitted
}
```

/* An observability is a query-method that is used to provide state information about the IUT to the TRM. That is, they are read-only methods that acquire and return a value stored by the IUT. */

```
Observability Integer
MarkForStudent(tStudent student);
Invariant IsFullCheck
{ Students().Length() <= CapSize(); }
```

/* Here is a simple responsibility, which will be used if prerequisites are set to not be enforced. It just checks that the student is not already in the course.
The keyword *Execute* indicates where execution occurs. */

```
Responsibility AddStudentNoPreReqCheck(tStudent s)
{  Pre(Students().Contains(s) == false);
Execute();
Post(Students().Contains(s) == true);   }
```

/* Other responsibilities include AddStudentPreReqCheck, RemoveStudent, new, finalize... see [2, 11] */

```
// a scenario defines a grammar of responsibilities
Scenario ReportMarks
{  Trigger(observe(TermEnded)),
once Scalar Contract University u = instance;
each(Students())
//built-in variable iterator accesses students one at a time
{u.ReportMark(context, iterator, MarkForStudent(iterator))},
Terminate(fire(MarksRecorded));     }
Exports
{ Type tStudent conforms Student { University::tStudent; }
}}} //end of Course contract
```

/* The Student contract represents an individual student enrolled at a university who is able to take courses. */

```
Namespace Examples.School {
Contract Student  {
//lots of observabilities and responsibilities are omitted.
```

/* We include the two scenarios of this contract to illustrate the semantic complexity our VF can currently handle. The keyword *atomic* defines a grammar of responsibilities such that *no* other responsibilities of this contract instance are allowed to execute except the ones specified within the grammar. We leave it to the reader to either figure out the details or read them elsewhere [11]. */

```
Scenario RegisterForCourses {
Scalar tCourse course;     Contract University u = instance;
Trigger(observe(CoursesCreated), IsCreated()),
```

/*triggered IF courses have been created and the student is also created (isCreated is a responsibility in this contract) */

```
 choice(IsFullTime()) true
  {  ( atomic
  { course = SelectCourse(u.Courses()),
    //via bindpoint get the instance for the selected course
    choice(course.bindpoint.IsFull())  true
      {  course = SelectCourse(u.Courses()),
        redo}   //until a course is not full
  },
// keyword context refers to the current contract instance
  u.RegisterStudentForCourse(context, course),
  RegisterCourse(course)
 )  [0-u.Parameters.MaxCoursesForFTStudents]
   //repeat up to the max # of courses for a full time
  }
   alternative(false)  //student is part-time (PT)
{ ( atomic
   {  course = SelectCourse(u.Courses()),
      choice(course.bindpoint.IsFull())  true
       { course = SelectCourse(u.Courses()),
         redo   }
     },
   u.RegisterStudentForCourse(context, course),
   RegisterCourse(course)
   )[0-u.Parameters.MaxCoursesForPTStudents]
 },
Terminate();  } //end of scenario RegisterForCourses

Scenario TakeCourses {
failures = 0; //number of failures in the current term
Trigger(observe(TermStarted)),
parallel
{ // for all courses of that term
 Contract Course course = instance;
  //if and only if that course is one taken by this student
  Check(CurrentCourses().Contains(course.bindpoint));
  atomic
   { (parallel//can do assignmnts, midterm, proj concurrently
       { (DoAssignment(course.bindpoint))
      [course.Parameters.NumAssignments]  }
   | //use OR, not AND, to avoid ordering
     (DoMidterm(course.bindpoint))
      [course.Parameters.NumMidterms]
   |
       (DoProject(course.bindpoint))
       [course sameas ProjectCourse &&
          course.Parameters.HasProject]
    ),
    (DoFinal(course.bindpoint)
      [course.Parameters.HasFinal]
     }
  alternative( not observe(LastDayToDrop))
    { DropCourse(course.bindpoint) }
}[CurrentCourses().Length()];
Terminate();  } //end of scenario TakeCourses
```

/* We omit most of the University contract, which does not add to this presentation. */

```
MainContract University
 { Parameters {
[1-100] Scalar Integer InstanceBind UniversityCourses; }
```

```
/* several parameters, observabilities, responsibilities and
some scenarios were omitted. */
Observability List tCourse Courses();
Observability List tStudent Students();
Responsibility ReportMark
/* The course, the student and the mark to be recorded are
provided as parameters. Each parameter of the responsibility
is bound to a parameter of the procedure bound to this
responsibility. */
   (tCourse course, tStudent student, Integer mark)
// the number of failures is recorded
 {  choice(mark) < Parameters.PassRate
       {  student.bindpoint.failures =
            student.bindpoint.failures + 1;  }  }
Responsibility CalculatePassFail()  {
 each(Students())
    choice(iterator.bindpoint.failures) >= 2
   FailStudent(iterator);
   alternative
   PassStudent(iterator);
Scenario CreateCourses
 { Trigger(new()),
 /* all courses of the term, in Parameters.UniversityCourses,
must be created */
   CreateCourse(dontcare, dontcare)
            [Parameters.UniversityCourses],
    Terminate(fire(CoursesCreated));  }
Scenario CreateStudents
 { Trigger(new()),
   CreateStudent(dontcare)+,
   Terminate(finalize());  }

Scenario Term
/* term management via doing responsibilities in a particular
order and firing the corresponding events */
 { Trigger(new()),
  ( CreateCourse()[Parameters.UniversityCourses],
    TermStarted(),
     fire(TermStarted),
     LastDayToDrop(),
     fire(LastDayToDrop),
     TermEnded(),
     fire(TermEnded),
     observe(MarksRecorded)
  [Parameters.UniversityCourses],
     CalculatePassFail(),
     DestroyCourse()[Parameters.UniversityCourses],
     fire(TermComplete)
   )+,
  Terminate(finalize());
 }
```

4 Scenario Testing with the VF

In ACL, scenarios are expressed as *grammars of responsibilities*, much like in Use Cases [5] and Use Case Maps [7]. As with any sort of grammar, there are well-known algorithms (e.g., [18, 19, 20]) to obtain a selection of paths through a grammar of responsibilities according to some *coverage criterion* [3]. In essence, a scenario (or responsibility) is parsed and transformed into a form of control flow graph [*Ibid.*] from which paths are easily extractable. However, such paths are not executable (and thus are often referred to as *test purposes*). As in several other methods, the VF currently supports the 'all branches' coverage criterion. Consider, for example:

```
Scenario X
{   Trigger (observe(eventA)) ,
    choice(failures) >= 2     responsibilityA();
      alternative             responsibilityB();
Terminate (fire(eventB));    }
```

Here, two branches need to be covered: one for 2 or more failures, one for less than 2 failures. In order to control this branching, a *path sensitization variable* (PSV) is required leading to two test cases: one for which the value of the PSV is set to 2 or more, another with a value less than 2. Currently, this form of *equivalence partitioning* [3] requires the user to set the actual PSV value used for each test case (for it cannot be inferred by the tool whether, for example, -1 or 0 are valid or not, and what is the maximum valid value for this PSV). It must also be pointed out that only valid PSV values are relevant for testing a scenario, as will be explained shortly.

As Binder explains at length [3], coverage of loops requires that they be flattened (i.e., 'unrolled'). So scenario **ReportMarks** in the *Course* contract will require a PSV to control the generation of the different paths associated with this loop. This PSV corresponds to the number of students in the course at hand. Currently, using the VF, minimally two test cases are generated: one for the minimum number of iterations and one for the maximum. If possible, a third test case for a number of iterations between this minimum and maximum is also generated. Thus, for the loop in **ReportMarks**, three test cases will be generated: one for a course with a minimum number of students, one for a course with the maximum number of students this course allows (i.e., its capsize), and one for a course with a number of students between this minimum and maximum.

Transforming a scenario into a graph from which paths can be extracted is not necessarily trivial as studying scenarios **RegisterForCourses** and **TakeCourses** in contract *Student* should make clear: combining branching statements (e.g., choice/alternative) with one another and with loop statements (e.g., redo and [] blocks) leads to complex control structures. The introduction of possible concurrent paths (through the *parallel* statement) further complicates this task. But, as previously mentioned, path generation is a well-understood process [18, 19, 20] for which we merely reuse existing solutions (by adapting them to the syntax and semantics of ACL). Conversely, the identification of PSVs requires an ACL-specific solution, which can be discussed only after we first understand what the VF offers in terms of support for scenario testing.

As hinted in section 2, the primary role of the VF is to monitor the execution of an IUT and report on violations of static and dynamic checks (such as violations of pre- and post-conditions of responsibilities, of invariants of contracts, and of grammars of scenarios). Focusing specifically in this paper on scenarios, we remark that an instance of a contract is created each time an instance of the type to which the contract has been bound is made. So, for example, each instance of a course created during the execution of the IUT is monitored by a corresponding instance of the *Course* contract. And each contract instance creates an instance of one of its scenarios once this scenario is triggered. So, for example, once the term ends, each course contract instance will create its scenario instance for scenario **ReportMarks** in order to monitor the grammar of that scenario for that specific course. A scenario violation will occur, for example, in any course for which its scenario instance for scenario **ReportMarks** fails to observe the responsibility *ReportMark*() of the university being called for the exact number of students in the course at hand. A scenario violation will also be recorded if the execution of the IUT terminates without a scenario having its *Terminate* condition satisfied.

The more complex the semantics of a scenario, the more complex its grammar and the more numerous its sources of violations. Consider, for example, scenario **TakeCourses** in contract *Student*. The key observation is that there is a single instance of this scenario for each student. Thus, this single scenario instance addresses the completion of *all* the courses taken by this student in that term. To do so, it verifies (amongst other checks) that the exact number of assignments for course *c* is performed (by tracking how many times the *DoAssignment*() responsibility is invoked with *c* as parameter). Should the student not complete the required number of assignments (or midterms, etc.) in any of her courses during the term at hand, then the VF will report a violation for scenario **TakeCourses**.

Most importantly, it is crucial to understand that, because responsibilities found in ACL contracts are bound to actual procedures of an IUT, scenario validation using the VF ensures that actual sequences of procedure calls (monitored during the execution of an IUT) 'obey' the grammar of the scenario(s) relevant to these procedures. With respect to scenario testing, this modus operandi of the VF defines what is *observable* [3]. But scenario testing requires that we address not only observability but also controllability [*Ibid.*]. To do so, let us return to scenario **ReportMarks**. As previously mentioned, testing this scenario involves one PSV for the flattening of the *each* statement. **ReportMarks also** invokes responsibility *ReportMark*() in the *University* contract. This further complicates PSV identification, as discussed at the end of this section.

For now, the immediate question is how test cases generated for a scenario are to be executed. ACL is an implementation-independent specification language and thus executable code cannot be generated from it. However, because ACL contracts are bound to classes and ACL observabilities and responsibilities to procedures, ACL scenarios can be used to monitor the correct execution of specific test cases. Let us elaborate on this by continuing our discussion of the testing of scenario **ReportMarks**.

In the context of testing the example university system, the execution of a corresponding IUT will involve the execution of a test suite, that is, of a set of test cases. The task of creating this test suite lies with the developer/tester: the role of the VF is to generate what we call *test scenarios* that address the coverage of the ACL contracts (not IUT code!) modeling the university system. Specifically, for scenario **ReportMarks**, this involves the following steps:
1) this scenario is selected in the testing window of the VF. (The user chooses which scenarios to test in a particular execution of the IUT.)
2) the scenario is parsed and the user is asked to input a name (e.g., *numberOfStudents*) for the PSV required to flatten the loop of the scenario (and for other identified PSVs, if any).
3) the user is prompted to input a minimum and a maximum default value for *numberOfStudents*.
4) the user may flag the observability MarkForStudent as a 'IUTProvided', in which case it is understood marks for students will be supplied via the execution of the IUT. Alternatively, the user may flag this observability as 'toInput', in which case each mark will have to be input by the tester at run-time. (This alternative is useful when wanting to avoid writing a multitude of similar test cases differing only with respect to these marks.)
5) From the information above, the VF generates a test scenario for the minimum value of *numberOfStudents* and another for its maximum value. If an in-between value is possible, the VF generates a third tes scenario for this value. (Examples of such test scenarios are discussed shortly.)
6) As the IUT executes, beyond the monitoring offered by scenario **ReportMarks**, the VF also monitors the generated test scenarios. A test scenario is 'covered' if it is triggered and terminates correctly. That is, whereas scenarios are monitored for violations, in the case of test scenarios, it is their occurrence at least once during the execution of an IUT that is reported to the user. (Thus, once the occurrence of a test scenario has been detected, the VF prevents this test scenario from being triggered again.)
7) At the end of the execution of an IUT, the VF reports on the occurrence or absence of each test scenario. The user can assess how much coverage of each one the scenarios has been achieved and add more test cases to the IUT where need be (in order to have more test scenarios covered).

The nature of a test scenario is best understood through a simple example. Consider the case for which scenario **ReportMarks** is to be tested with a minimum number of students. The generated test scenario is:

```
TestScenario ReportMarks-1
{  Trigger(observe(TermEnded)),
once Scalar Contract University u = instance;
```

```
Check(Students.Length() == numberOfStudents.Min());
Terminate(fire(MarksRecorded));     }
```

This test scenario will have been covered once any empty course completes. Now consider the case for which scenario **ReportMarks** is to be tested with a maximum number of students. The self-explanatory generated test scenario is:

```
TestScenario ReportMarks-2
{  Trigger(observe(TermEnded)),
once Scalar Contract University u = instance;
Check(Students.Length() == numberOfStudents.Max());
Terminate(fire(MarksRecorded));
```

Should the tester want to use a maximum number of students specific to each course, then the appropriate Check statement would be:

```
Check(Students.Length() == context.CapSize());
```

While these simple examples summarize the role of generated test scenarios with respect to scenario coverage, they do not convey the complexity of i) PSV identification and ii) generation in more complex scenarios. We discuss these issues next.

Consider scenario **RegisterForCourses** in contract *Student*. Whether a student is full-time or part-time leads to two different *sets* of generated test scenarios. For each of these two sets, there are several other PSVs:

- the number of courses successfully registered in (required to flatten out the statement [0-u.Parameters.MaxCoursesForFTStudents] for full-time student (or its equivalent for part-timers).

Here the minimum and maximum values are explicitly captured and need not be asked from the user.

- the maximum number of iterations of the redo statement, that is how many courses that are full can be selected before one non-full one is found. Because of the semantics of the redo, the minimum is implicit: if the first course selected is available, the redo does not execute.

Given these three PSVs, the VF will generate test scenarios corresponding to different combinations of values for these PSVs. For example:

- a full-time (or part-time) student who does not attempt to register in any course
- a full-time (or part-time) student who registers in the maximum allowable number of courses for her status, all these courses being immediately available (i.e., non full and thus no redo is performed).
- a full-time (or part-time) student who registers in the maximum allowable number of courses for her status, each of these available courses being selected only after the maximum number of retries (i.e., full courses) has been attempted.

In summary, a scenario can have several PSVs associated with it and (via boundary analysis [3] on these PSVs) a test scenario will be generated for each possible valid combination of PSV values. (Representations and algorithms for such combinations are discussed at length in chapter 6 of [3].) Due to the semantic richness of ACL, such combinatorial testing can be quite complex. Consider, for example, scenario **TakeCourses** in contract *Student*. The *parallel* statement and or (|) operators used in the atomic block allow for the requirements of a course to be addressed in *any* order as previously mentioned. And, for assignments and midterms, loops are involved and must be flattened (thus requiring a min/max for the number of assignments and the number of midterms). In contrast, for the possible project and final, the choice is Boolean: a course has 0 or 1 project and 0 or 1 final. What further complicates the corresponding control flow graph is the *parallel* statement used at the start of the scenario to allow a single student to have the requirements of *several* courses taken the same term be addressed concurrently. Whereas a loop requires one PSV, a *parallel* statement requires two: i) the min/max number of instances (defined in this scenario by CurrentCourses().Length()) and ii) the min/max number of concurrent instances. That is, for this scenario to be thoroughly covered, we need to know not only the maximum number of courses a student can take in a term but also how many of these courses can have their requirements be concurrent. Only with these two PSVs can we test not only courses with simultaneous requirements, but also courses whose requirements do *not* overlap in time (i.e., minimum concurrency is set to 0). A last PSV (capturing the min/max number of courses dropped) is required to control whether or not courses are droppedduring a term.

Three observations proceed from this example:

1) Test scenarios are *not* meant to address issues such as trying to drop a course after the last day to drop: the scenario itself will catch such violations.

2) It should be clear that not all paths are covered: there is a combinatorial explosion of possible paths, especially in light of the interleaving [18] resulting from the use of *parallel* statements. This is why we rely on an existing algorithm for 'all-branches' coverage [3, 18].

3) The scope of a PSV is a test scenario. Thus there is no way in **TakeCourses** to associate a maximum number of assignments to a specific course. Semantically this could be desirable but would not only greatly complicate the generation of combinations of PSV values, but also dramatically decrease the usability of our testing approach (as a multitude of user inputs would be required).

The fact that a scenario invokes one or more responsibilities constitutes another source of complexity in identifying PSVs. Let us return to scenario **ReportMarks** of the *Course* contract. It invokes responsibility *ReportMark*() of the *University* contract. We postulate that, in the context of testing **ReportMarks**, both branches (i.e., a failing grade or, implicitly, a passing one) of *ReportMark*() must be covered (regardless of unit testing on the procedure bound to this responsibility). The PSV to control this is generated by the VF in the scope **ReportMarks**. Its name summarizes best its semantics: *numberOfFailingMarks* (over the scenario's execution). As usual, it is left to the user to specify the min and max for this PSV (from which the system will generate

an in-between value, if possible). Once, this is done, test scenario generation can proceed: the loop of the scenario leads to 2 or 3 paths (as explained earlier), each of which now branching into one of the 2 or 3 paths associated with *numberOfFailingMarks*. Thus, the VF generates a minimum of 4 test scenarios for **ReportMarks**: (min # of students, min # of failures), (min # of students, max # of failures), (max # of students, min # of failures) and (max # of students, max # of failures).

There two points to make here: i) and ii) the generation of test scenarios does not merely proceed from generating combination of PSV values; it also involves the flow analysis of the scenario (and the responsibilities it invokes). Thus, in summary, the generation and validation of test scenarios in ACL is a complex task relying on user input for setting up correctly boundary value testing.

5 Conclusion

We have presented elsewhere i) ACL, a semantically rich implementation independent specification language (that supports design by contract, responsibilities and scenarios) and ii) the tool (the VF) that enables its user to bind ACL specifications to the types and procedures of an IUT, thus enabling the validation of an ACL against an actual execution of an IUT. In this paper we have focused specifically on scenario testing and we have overviewed how ACL specifications can be used to generated test scenarios and their path sensitization data. Most importantly, using the VF, these test scenarios can be validated against an actual execution of an IUT. We believe only Spec Explorer [8] offers similar semantic richness and scenario testing capabilities, albeit using the symbolic execution of a complex finite state machine.

Acknowledgments

We thank Katie McClean for her implementation of the ACL parsing required to generate the PSVs. And thanks to NSERC!

6 References

[1] B. Meyer, The Unspoken Revolution in Software Engineering, *IEEE Computer*, January 2006.

[2] D. Arnold, J.-P. Corriveau and W. Shi, *Reconciling Offshore Outscouring with Model-Based Testing*, *SEAFOOD*, Saint Peterburg, Russia, June 2010.

[3] R., Binder, *Testing Object-Oriented Systems*, Addison-Wesley Professional, Reading, MA, 2000.

[4] J. Ryser and M. Glinz. *SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test*. Technical Report. University of Zurich, 2003.

[5] D. Rosenberg and M. Stephens, *Use Case Driven Object Modeling with UMLTheory and Practice*, APress, 2007.

[6] Message Sequence Charts, http://www.itu.int/ITU-T/2005-2008/com17/languages/Z120.pdf

[7] Buhr, R.J.A., Casselman, R.: *Use Case Maps for Object Oriented Systems*. Prentice Hall, November 1995.

[8] C. Campbell, W., Grieskamp, L., Nachmanson, W., Schulte, N., Tillmann, and M.Veanes. *Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer*. Microsoft Research Technical Report #MSR-TR-2005-59, May 2005.

[9] Microsoft Research: *Spec# Tool*. http://research.microsoft.com/specsharp

[10] D. Arnold, J.-P. Corriveau and W. Shi, *Validation against Actual Behavior: Still a Challenge for Testing Tools*, *SERP*, Las Vegas, July 2010

[11] D. Arnold, The Validation Framework and its examples, http://vf.davearnold.ca/.

[12] IBM: *Rational Rose*. http://www-306.ibm.com/software/awdtools/developer/rose/index.html

[13] IBM: Rational Robot, http://www-01.ibm.com/software/awdtools/tester/robot/

[14] JUnit, http://www.junit.org/

[15] B. Meyer et al., Programs that test themselves, *IEEE Computer*, vol.42(9), September 2009, pp.46-55.

[16] B. Meyer, Design by Contract. In *IEEE Computer*, vol. 25, no. 10, pp. 40-51, IEEE Press, New York, October 1992.

[17] D. Arnold, J.-P. Corriveau and W. Shi,, Modeling and Validating Requirements using Executable Contracts and Scenarios, *SERA*, Montreal, May 2010.

[18] L. Briand and Y. Labiche, *A UML-Based Approach to System Testing*, Lecture Notes In Computer Science; Vol. 2185, Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools, 194 - 208

[19] C. Nebut, F. Fleury, Y. Le Traon J.M. Jézéquel. *Automatic Test Generation: A Use Case Driven Approach*. *IEEE Transactions on Software Engineering* Vol. 32, 2006

[20] A. Miga, Applications of Use Case Maps to System Design with Tool Support, M.Eng. Thesis, Dept. of Systems and Computer. Engineering, Carleton University, 1998.

Performance Evaluation of Testing for Maintaining Software-Quality

Rajat Sheel Jain[1], Dr. Amit Gupta[2]

Department of Information Technology, Institute of Management Studies, Noida, India

[1]*jainrajatsheel@gmail.com*

Maharaja Agrasen Institute of Management Studies, New Delhi, India

[2]*amitgupta21@gmail.com*

Abstract - Test development is an expensive technique. Saving the test suite for the software application by which test cases from the suite can be used for the software maintenance. We propose to develop Specification Analyser that accepts specification like statement coverage, code coverage for the generation of efficient test cases from test suite. The Specification Analyser compares the information about the techniques like Precision, Efficiency, Inclusiveness and Generality. The Fault detection capability tool provides the minimized size of test suite which satisfies the above criteria. By reducing the test suite size, we can reduce the execution cost and time, validation and management of the test cases from the suite for future releases of the software and able to maintain the fault detection capability by reusing the refined test cases. The prioritization method will increase time-effectiveness in detecting the faults. An improved rate of fault detection can provide faster feedback of the system under test.

Keywords- Regression Testing, Regression Test Selection, Linear Equation, Symbolic Equation, Dataflow Techniques, Prioritisation

I. INTRODUCTION

Software Testing is the activity that individual does with the intention to find out the errors in software applications. Regression Testing is the process of validating modified software to detect whether the new errors have been introduced into the previously tested codes and provide confidence that the modifications are correct.

Since the regression testing is an expensive process, researches have proposed regression test selection techniques as a way to reduce some of this experience. These techniques attempt to reduce the costs by selecting and running subsets of the test cases in the program's existing test suites. However it is difficult to compare and evaluate these techniques because they can be used to solve the different problem goals.

A typical selective retest technique proceeds as follows:

1) Select $T^1 \subseteq T$, a set of tests to execute on P^1.

2) Test P^1 with T^1, to establish the correctness of P^1 with respect to T^1.

3) If necessary, create T^2, a set of new functional or structural tests for P^2.

4) Test P^1 with T^2, to establish the correctness of P^1 with respect to T^2.

5) Create T^3, a new test suite and test history for P^1, from T, T^1 and T^2.

All code-based regression test selection techniques attempt to select a subset T^1 of T that will be helpful in establishing confidence that P^1 was modified correctly and that P's functionality has been preserved where required. In this sense, all code-based test selection techniques are concerned, among other things, with locating tests in T that expose faults in P^1. Thus, it is appropriate to evaluate the relative abilities of the techniques to choose tests from T that detect faults.

II. REGRESSION TEST SELECTION TECHNIQUES

There are four categories for the regression test selection techniques: Linear Equation, Symbolic Equation, Path Analysis Technique, Data flow Techniques.

A. Linear Equation Techniques

A selective retest technique that uses systems of linear equations to select test suites that yield segment coverage of modified code. Linear equation techniques use systems of linear equations to express relationships between tests and program segments. The techniques obtain systems of equations from matrices that track program segments reached by test cases, segments reachable from other segments, and (optionally) definition-use information about the segments.

Linear equation techniques are automated. When the techniques operate as minimization techniques, they return small test suites and thus reduce the time required to run the selected tests. However, Fischer states that due to the calculations required to solve systems of linear equations the techniques may be data and computation intensive on large programs. In fact, the underlying problem is NP-hard, and all known 0-l integer programming algorithms may take exponential time. Despite this possible worst-case behaviour, 0- 1 integer programming algorithms exist that can obtain solutions, in practice, in times that may be acceptable.

Both intra-procedural and inter-procedural techniques require computation of a correspondence between segments in P to segments in P', and of which segments have been modified, after testing has entered its critical phase. The references on linear equation methods do not specify a method by which correspondence and change information should be computed.

B. Symbolic Execution Techniques

A selective retest technique that uses input partitions and data-driven symbolic execution to select and execute regression tests. Initially, the technique analyses code and specifications to derive the input partition for a modified program. Next, the technique eliminates obsolete tests, and generates new tests to ensure that each input partition class is exercised by at least one test. Given information on where code has been modified, the technique determines edges in the control flow graph for the new program from which modified code is reachable. The technique then performs data driven symbolic execution, using the symbolic execution tree to symbolically execute all tests. When tests are discovered to reach edges from which no modifications are reachable, they need not be executed further. Tests that reach modifications are symbolically executed to termination. The technique selects all tests that reach new or modified code. However, the technique also symbolically executes these selected tests, obviating the need for their further execution.

C. The Path Analysis Technique

There will be another technique for the selective retests that will be known as the Path Selective Analysis Technique. The technique takes as input the set of program paths in P' expressed as an algebraic expression, and manipulates that expression to obtain a set of cycle-free exemplar paths: acyclic paths from program entry to program exit. The technique then compares exemplar paths from P to exemplar paths from P', and classifies paths as new, modified, cancelled, or unmodified. Next, the technique analyses tests to see which exemplar paths they traverse in P. The technique selects all tests that traverse modified exemplar paths.

D. Data Flow Technique

Several selective retest techniques are based on dataflow analysis and testing techniques. Dataflow test selection techniques identify definition-use pairs that are new in, or modified for, P', and select tests that exercise these pairs. Some techniques also identify and select tests for definition use pairs that have been deleted from P. Two overall approaches have been suggested. Incremental techniques process a single change, select tests for that change, incrementally update dataflow information and test trace information, and then repeat the process for the next change. Non-incremental techniques process a multiply-changed program considering all modifications simultaneously.

III. PROPOSED SPECIFICATION ANALYSER APPROACH

In this paper we proposed to implement the Incremental algorithm which selects the test cases

from test suite T whose outputs may be affected by the modifications made to the programs.
The algorithm exploits the following observations:
1. Not all statements in the program are executed under all test cases.
2. If a statement is not executed under a test case, it cannot affect the program output for that test case.
3. If a statement is executed under a test case, it does not necessarily affect the program output for that test case
4. Every statement does not affect every part of the program output.

The specification analyser will work as a parser. The test cases from the test suite pass to the analyser where these cases will be observed in terms of the statement coverage and costs-benefits.

Applying the algorithm to parse which decomposes the program and selects test cases to ensure that there is no linkage between the modified and unmodified code.

IV.SYSTEM ARCHITECTURE

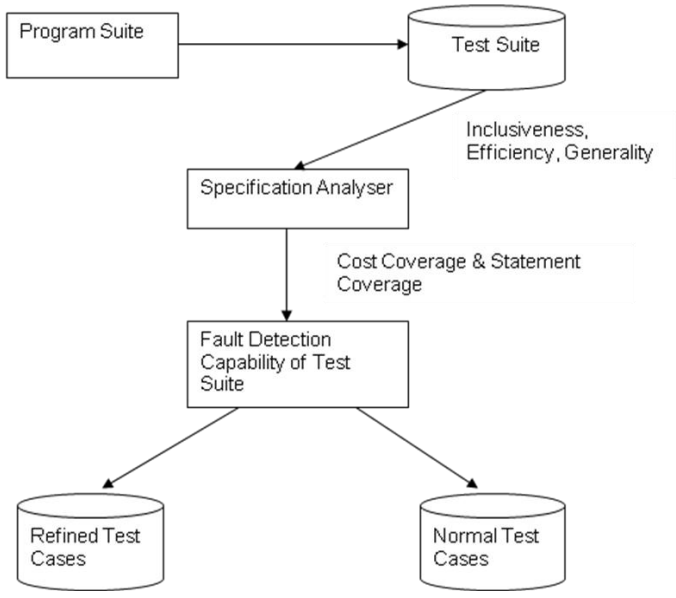


Fig. 1 System Architecture

The above framework will help us to refine the test suite and measure its fault detection capability. There will be the source program through which we can take the input and give it to the test suite.

The test suite that contains the numbers of the test cases specifies that which test cases will cover more number of the program to satisfy desired criteria.

The size of the test suite minimizes without reducing their fault detection capability of the test cases in the suite.
The framework analyses the test cases on the basis of the four properties i.e. Inclusiveness, Precision, Generality and Efficiency. Inclusiveness measures the extent to which a technique chooses tests that will cause the modified program to produce different output than the original program, and thereby expose faults caused by modifications. Precision measures the ability of a technique to avoid choosing tests that will not cause the modified program to produce different output than the original program. Efficiency measures the computational cost, and thus, practicality, of a technique. Generality measures the ability of a technique to handle realistic and diverse language constructs, arbitrarily complex code modifications, and realistic testing applications. These categories form a framework for evaluation of the test cases that should be analysed through our specification analyser and compare them.

V. RESULTS AND DISCUSSIONS

The fault detection capability tool may refine those test cases that perform for the maximum statement coverage and having the minimum cost. The cost could be found on the basis of the fact that, suppose that there are hundreds of test cases in a test pool and out of which only 30% test cases are found that satisfy the criteria for the statement coverage and their fault detection capability or efficiency cannot be affected due to any modifications made to the program.

The main benefit of our framework is to evaluate each and every test case from the test suite on the given criteria basis and compare them with the manual test cases that can be directly given to the tool. The tool helps us to evaluate which test cases are more efficient: either manual test cases or those that would be analysed by the specification analyser through the different properties.

The fault detection capability tool may reduce the test suite sizes by refining them but the minimization cannot compromise the fault detection effectiveness of the coverage of the statements.

VI. CONCLUSION AND FUTURE WORK

This paper focuses that the proposed frame work is used to identify the strengths and the weaknesses of those test cases that cannot help for the minimization of the test pool and may affect their fault detection capability. With our framework we have to analytically evaluate the fault detecting capability of the test suite and efficiency in terms of cost and time for that test suite.

Our evaluation indicates that despite of different goals of the various properties, the effectiveness of the fault detection can be compared and understood if our framework is used for the industrial purposes.

Once the framework is to be applied it is used to demonstrate that for a given test pool how many test cases are efficient for the finding of the statement coverage and how much cost is to be obtained for these test cases.

We proposed a pre-emptive designing "Evaluation of the Fault Detection Capability of the Test Suite" for the future work on the given hypotheses:
(a) How the minimization is fair in terms of costs and benefits when the coverage of test suite is adequate?
(b) If the test suite size increases then how the fault detection effectiveness should be affected?

ACKNOWLEDGMENT

One of the authors (Rajat Sheel Jain) express his sincere gratefulness to Mr. Rajeev Kumar Gupta, President and Mr. Alok Agrawal, Advisor, Institute of Management Studies, Noida, India, for their encouragement and support throughout the work of this wish and also for facilitating technical and literature facilities, required in the development of this work. He is also expresses his thanks to Dr. Vijaypal Dhaka, Associate Professor & Head, Department of IT, Institute of Management Studies, Noida, India for his valuable suggestions.

REFERENCES

[1] G. Rothermel and M. Harrold. Analyzing regression test selection techniques. *IEEE Trans. On Softw. Eng.*, 22(8):529-551, Aug. 1996.

[2] W. E. Wong, J. R. Horgan, A. P. Mathur, and A. Pasquini. Test set size minimization and fault detection effectiveness: A case study in a space application. In *Proc. of the 21st Annual Int'l. Comp. Softw. & Appl. Conf.*, pages 522-528, Aug. 1997.

[3]. Agrawal, H., Horgan, J.R., Krauser, E.W., and London, S.A. Incremental regression testing. In *Proceedings of the IEEE Software Maintenance Conference* (1993), pp. 348–357.

[4]. Rothermel, G. and Harrold, M.J. *A Comparison of Regression Test Selection Techniques.* Tech. Rep., Department of Computer Science, Clemson University, Clemson, SC, Oct. 1994.

[5] J.-M. Kim, A. Porter, and G. Rothermel. An empirical study of regression test application frequency. In *Proc. of the 22nd Int'l. Conf. on Softw. Eng.*, June 2000.

[6] D. Rosenblum and G. Rothermel. A comparative study of regression test selection techniques. In *Proc. of the 2nd Int'l. Workshop on Empir. Studies of Softw. Maint.*, Oct. 1997.

[7]. Rothermel, G. and Harrold, M.J. A safe, efficient algorithm for regression test selection. In *Proceedings of the IEEE Software Maintenance Conference* (1993), pp. 358–367.

[8] W.E. Wong, J.R.Horgan, S.London, and A.P.Mathur ,"Effect of the Test Set Minimization on Fault Detection Effectiveness,"Proc.17th Int'l Conf.Software Eng., pp.41-50, Apr.1995.

[9] G. Rothermel and M.J. Harrold, "Selecting Tests and Identifying Test Coverage Requirements for modified Software," *Proc. 1994 Int"Symp. Software Testing and Analysis*, pp.169-184, Aug. 1994.

Evaluation of the Testing Methods in Agent-Oriented Software Engineering

A. Saeed Zamani [1], B. Ramin Nassiri [2] and C. Sam Jabbehdari [1]
[1]Department of computer engineering, Islamic Azad University, Tehran north branch, Iran
[2]Department of computer engineering, Islamic Azad University, Tehran central branch, Iran

Abstract - *Testing is an important process that can assure the quality and the correct functionality of the multi agent systems (MAS). Multiple testing methods in agent-oriented software engineering (AOSE) have been introduced in recent years. Although the quality of the system is dependent on the quality of the applied test method, very little attention has been paid to evaluating these testing methods. As a result, it is difficult to select a sufficient method for testing an agent-based system. Additionally, there are no means to determine what the advantages and drawbacks of each method are. This paper proposes a framework for evaluating and comparing the testing methods in AOSE. This framework addresses major divisions of a testing method. The framework is then used to evaluate some prominent testing methods, have been proposed so far. A subset of these testing methods, which cover more criteria in the proposed framework, is presented.*

Keywords: Agent-Oriented Software Engineering, Testing Methods, Evaluation Framework, Comparing the Test Methods.

1 Introduction

Agent-Oriented Software Engineering (AOSE), is concerned with how to specify, design, implement, verify (including testing and debugging), and maintain agent systems [36]. The objective of AOSE is to efficiently and effectively develop high-quality agent-based software products. Nowadays, intelligent agent-based systems are applied to many domains including robotics, network security, traffic control, and ecommerce. Therefore, the owners and the operators of these systems need guarantees over quality and correct functionality of them [11]. This calls for suitable software engineering frameworks, including testing techniques, to provide high-quality software development processes and products [23].

During the last decade, many methodologies have been proposed for developing agent-based systems but current state of AOSE paradigm reports relative lack of industrial acceptance [1]. Furthermore, the application of these methodologies is still limited due to their lack of maturity and standardization. Testing is one of the most urgent activities that are often disregarded in most agent-oriented methodologies [7]; mainly because they focus on analysis and design activities, and relegate the implementation and testing to the traditional techniques [20]. Software testing is one of the most important phases in software engineering, and plays a pivotal role in software quality assurance.

Under ideal situations, with minimal testing efforts, integration of reliable software agents should produce high-quality agent-based systems. In reality, however, many agent-based software characteristics, such as autonomy, pro-activity, mutual relationships of these agents and relationships with the environment impose great difficulties on achieving this goal and make traditional techniques inefficient.

To thoroughly understand the difficulties and key issues in testing and maintaining the agent-based software, and thereby to apply adequate methods, this paper focuses on the following questions:

1. What are the key characteristics of agent-based systems that distinguish them from other systems (merely according to testing)?
2. How can agent oriented software testing methods verify these characteristics?

In order to answer these questions, the testing issues are characterized by proposing a framework to evaluate the existing testing methods. We consider the methods employed by agent-oriented methodologies (there are also several methodologies that do not include testing in their process models [7]) and the methods that are not related to any specific methodology (e.g. [9]). Comparing prominent agent-oriented testing methods and evaluating their strengths and weaknesses, play an important role in improving their performance. This can also contribute to applying appropriate testing methods or combinations of various methods and techniques. Within the last few years many frameworks have been proposed for evaluating AOSE methodologies, e.g. [17], [35]. These frameworks merely check if the testing process is mentioned in the methodologies or not. There is no work on verifying the quality of the testing methods in AOSE methodologies. Yet, comparing methods is often difficult, because they might address different aspects or differ in their terminology. For instance, some methods verify the static structure of the agent systems [4], [23], while the others verify the dynamic behavior [9]; some focus on the *Agent Level* of the systems while some consider that the agents are reliable and focus on the *Society Level* of the

systems. Comparing is also problematic with some methods that are influenced by a specific methodology (e.g. *Tropos*, *MaSE* and *Prometheus*).

This paper is organized as follows. Section 2 describes our proposed framework for comparing and evaluating AOSE testing methods. In Section 3 the framework is applied in order to compare the existing multi-agent testing methods and finally Section 4 concludes the paper.

2 The Evaluation Framework

In [11], six testing method were evaluated. The evaluation criteria are divided into two test levels and their test types (*white box* and *black box*). In this paper, a comprehensive framework of evaluating and comparing agent-oriented testing methods is proposed. This framework offers a well-defined, structured set of aspects that an agent-oriented testing method should include. The first major division of the framework is based on the framework suggested by [2]. This study extends and modifies this framework to address the properties of a comprehensive testing method in AOSE. The other major divisions that are being inspired by [3] and [35] are not specifically related to AOSE and could be considered in other software engineering paradigms, e.g. *object oriented* and *procedural*. We refer to a testing method as the entire set of these major divisions:

- Multi-agent systems test basics
- Test process
- Test techniques
- Test pragmatism

Each of these four major divisions includes their specific criteria that will be explained in the following. The proposed framework is illustrated in figure 1.

We emphasize that these four divisions, are in fact four different views of the whole test method, and can overlap; i.e. some of the criteria may be present in different divisions, having different names but the same identity.

2.1 Multi Agent Systems Test Basics

Multi-agent systems testing are normally divided into multi layers [9], [11] and [29]. According to the *V-model* [31], the *Test Level*s are: testing agents as individual units of the MAS, testing the integration of collaborator agents and testing the whole system. *Dynamic Testing* (will be discussed in 2.3) should be used in the first layer, in order to verify the behavior of an agent. Much of this testing would require another agent to trigger an event inside the agent to be tested, such as a message from another agent, or an event from the environment [9]. Furthermore, *Static Testing* (will be discussed in 2.3) should be used for validation in the first layer.

We have changed the general V-model by adding a layer called "*Agent Acceptance Test*" after testing the functionality of an agent in the first layer. This test layer is concerned with the essential properties of agents such as *Autonomy*, *Pro-activity* and *Sociability*. Testing agents, according to these properties, is the most challenging task which makes the traditional testing methods insufficient.

Two main issues in *Integration Level* testing of an agent-based system are to be considered: (i) data models define the contents and format of the interactions in the control protocol and (ii) as agent-based systems are built under a distributed environment, which will then inherit all issues of the distributed systems, such as race conditions and deadlocks.

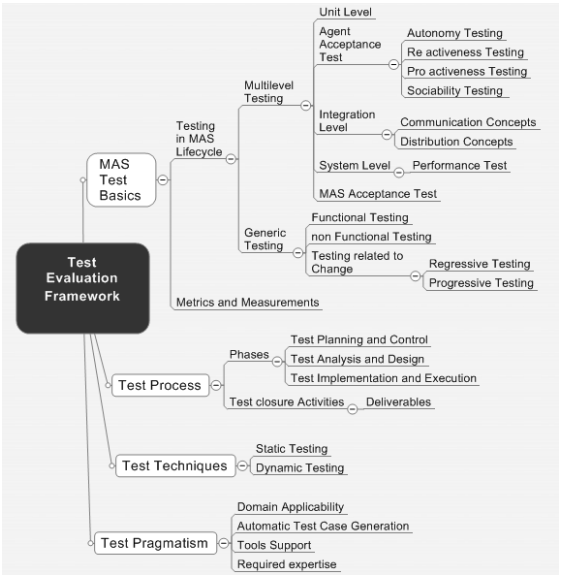


Figure 1 - Multi Agent Systems Test Evaluation Framework.

Agents are different from objects and interaction is based on communication language and protocols, rather than invoking the functions of each other. In addition, the agent-based systems include agents that *autonomously* pursue their individual goals and access resources and services of the environment. Therefore, deadlock detection techniques must differ from other distributed systems. A set of errors that could occur in unit level and integration level of agent-based system are presented in [28].

After the end of the *System Level* testing, the functionality of the whole system could be assured. Furthermore, *Performance Testing* is needed to verify that all of the worst case performance targets have been met (according to the resource constraints within the system, e.g., time, CPU and memory). Since these systems are also non-deterministic [11], this kind of testing will be challenging too. Within the last layer, *Validation* should be performed to find out whether the system has met the stakeholders requirements or not.

Testing types (e.g. *functional*, *nonfunctional* and *regression*) are independent of a particular test level. In

the *Generic Testing* division, the evaluation is about the existence of test types in the test levels. For instance, since agents are work flexibly in a dynamic environment without continuous direct supervision [11], and may change through the time, *regression testing* (for parts that have been changed) and *progressive testing* (for parts that have been added) must be performed in the *integration level*.

The last important factor in testing is how to define *Quality Metrics* in AOSE. Well-established metrics and measures, aligned with project objectives, will enable the tester to track and report the test and quality results. A lack of metrics and measurements leads to subjective assessments of quality and testing [3]. Not only metrics and measurements are crucial, but also *baselines* (An acceptable result), are required to verify the actual quality of the system under test, against the expected quality. Although these metrics are different from traditional software metrics, only few studies have addressed the issues of AOSE metrics.

2.2 Test Process

We should investigate the way that any testing method looks at the *Test Process* in the AOSE. If the method does not consist of phases then it more looks like an activity, rather than a process, and may delay testing until the end of implementation. According to *ISTQB* framework[1] [3], a test process consists of the following activities:

- Planning and control
- Analysis and design
- Implementation and execution
- Evaluating exit criteria and reporting
- Test closure activities

For any particular testing method, *Test Process criteria* involve clarifying what activities of a software testing process are mentioned within the agent-based system lifecycle.

Test Planning sets a framework for deriving *Test Cases*[2] and *Test Conditions* from the *Test Basis*. The test basis may include *requirements specifications*, *design specifications*, *quality risks*, and some other items. In the *Test Control*, the test method compares actual progress against the plan. The *Test Objectives* are a major deliverable. The *Test Analysis and Design* involves the following sub-activities:

- Identifying and refining the test conditions for each test objective.
- Creating test cases that exercise the identified test conditions.
- Creating *Test Oracles* (will be discussed in 2.3).

1 International Software Testing Quality Board

2 The comprehensive definitions of the aforementioned testing terms can be found in [12].

Test Implementation includes all the remaining tasks necessary to execute the test cases. In this activity, the test method should run a Single Test Procedure and log the Test Results. The *Evaluation of Exit Criteria* and reporting of results is a test management activity. Delivering test work products (e.g. error reports, test plan, etc.) is one of the *Test Closure Activities* that, a test method could have.

2.3 Test Techniques

There are two kinds of *Testing Techniques* that presented in the first level of the test techniques division [3]: (i) static testing (i.e. testing the system without running it) and (ii) dynamic testing (i.e. testing the system during its runtime). The test techniques are applied in the test types (presented in 2.1).

A sufficient testing method should cover both the static and the dynamic testing techniques. The input of the static testing could be the *AOSE Artifacts* that get developed in the agent-oriented methodologies [11]. Static and dynamic testing inputs are illustrated in figure 2. The *Model checking* approaches seem to be more acceptable static techniques, (because of having less complexity and better traceability [19]), since these methods propose that testing could be in some way based on the models of the system, which are abstractions of the actual system, and can be used for automated generation of test cases. Static testing has also the potential to lead to more accurate requirements *Verification*.

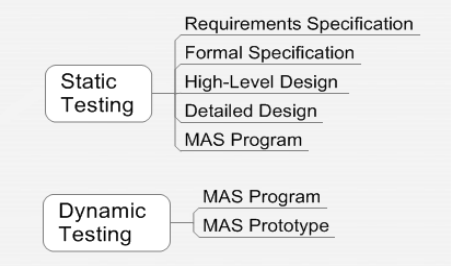


Figure 2- Input Artifacts for Static and Dynamic Testing.

On the other hand, dynamic testing is needed for validating the behavior of agents and the MAS as a whole. White box testing and black box testing are two common dynamic testing. White box testing can be performed in a traditional way, while there are some problems with the black box testing: It is very hard to find a test oracle (i.e. a source to determine expected results to compare with the actual result of the software under test [12]) for black box testing because of self-adaptation, learning and the autonomy of agents and successive tests with the same test data may give different results [28].

2.4 Test Pragmatism

In this division, we examine the practical aspects of using a test method within an agent oriented methodology. If the method is independent from a

particular methodology, then it becomes more applicable. Some methods that are evaluated in section 3 are proposed for specific agent architectures and agent-oriented methodologies. On the other hand, the methods that are proposed within AOSE methodologies seem more reliable. The main reason of the limited applicability of testing methods is that they are very challenging and expensive since it is quite complicated to automate them [6].

A sufficient testing method should propose a supporting tool and an automatic test case generator to reduce the time required for testing [27], and have *visualization* techniques, to become more acceptable.

These methods could also be evaluated on the mathematical sophistication level (e.g. exploiting the *Petri Nets*) and knowledge (e.g. *BDI* architecture and *Formal* methods) required to fully exploit the method. This consideration could enable an AOSE methodology to adopt a test method, better within its process.

2.5 Metric

In order to rank the properties examined in the evaluation process, we propose a scale of 1 to 3 as follows:

1: Indicates that the test method does not address the property.

2: Indicates that the test method refers to the property but not enough details are provided.

3: Indicates that the method addresses the property with a particular technique.

We emphasize that these numbers are not representing the quality of each property. They only indicate the existence of a property in the relevant methods; albeit with an exception for the *Test Pragmatism* that will be explained in section 3.

Table 1 – The Evaluation of the Test Methods in AOSE

The Evaluation of Test Methods in AOSE	Test Criteria	MAS Test Basics														Test Process				Test Techniques		Test Pragmatism			
Test Methods		Unit	Autonomy	Reactivity	Pro-activity	Sociability	Communication	Distribution	System	Performance	Acceptance	Metrics & Measurement	Functional	Non Functional	Change Related	Planning & Control	Analysis & Design	Implementation	Closure Activities	Static	Dynamic	Domain Applicability	Tool Support	Required Expertise	Automatic Test Case generation
[29] 3-layer		3	1	1	1	1	1	3	1	3	1	1	3	3	1	1	2	2	1	1	3	3	1	1	1
[18] Automated BDI		1	1	1	1	1	1	1	3	1	1	1	3	1	1	2	3	3	1	3	1	1	3	3	3
[32,33] Conversation Verification		3	1	1	1	1	3	3	1	1	1	1	3	3	1	1	3	3	2	3	3	1	3	3	3
[27] Design artifacts		3	1	1	1	1	3	1	3	1	2	1	3	1	1	2	3	3	3	3	3	3	3	3	3
[22] Evolutionary Testing		2	2	1	1	1	3	1	3	1	1	3	3	1	2	3	3	3	3	1	3	3	3	1	3
[23] Goal-Oriented		3	1	1	1	1	3	3	3	3	3	1	3	3	3	3	3	3	3	3	1	2	3	2	2
[13] INGENIAS		3	1	1	1	1	3	1	3	2	1	1	3	2	1	2	2	2	1	3	3	1	3	1	1
[8] JAT		3	1	1	1	1	3	1	1	1	1	1	3	1	1	2	3	3	2	1	3	3	3	1	2
[15] MadKit		3	1	1	1	1	3	1	3	3	1	1	3	3	3	1	1	3	1	1	3	3	3	1	1
[6] MAZBD		3	1	1	1	1	2	1	2	1	1	1	3	1	2	2	2	3	1	1	3	2	3	1	3
[9] Mock Agent		3	1	1	1	1	1	1	1	1	1	1	3	1	1	2	2	3	2	1	3	3	3	1	2
[19] Model-based Deadlock detection		1	1	1	1	1	1	3	1	1	1	1	1	3	1	3	3	3	2	3	3	2	3	2	2
[24,25] Ontology-Based		3	1	1	1	1	3	1	1	1	1	2	3	2	1	2	3	3	1	1	3	3	3	1	3
[26] Prometheus		1	1	1	1	1	3	1	3	1	1	1	3	1	1	2	2	2	2	1	3	1	3	3	1
[30] Regression Testing		3	1	1	1	1	3	1	1	1	1	1	3	1	3	1	2	1	1	1	3	3	1	1	1
[10] SEAUnit		3	1	1	1	1	3	1	3	2	2	1	3	3	2	1	1	2	1	3	3	3	3	1	1
[34] SUNIT		3	1	1	1	1	3	1	3	1	2	1	3	2	2	2	2	3	2	3	3	2	3	2	3
[28] Test Agent		3	1	1	1	1	3	3	2	1	1	3	3	3	3	1	2	3	1	3	3	3	2	1	2
[4] Verifying by model checking		1	1	1	1	1	1	1	3	1	1	1	3	2	1	1	1	3	2	3	1	2	3	2	1
[16] XP		3	1	1	1	1	3	1	2	1	1	1	3	1	1	1	2	2	1	1	3	1	2	1	2
[21] Zeus		3	1	1	1	1	3	1	3	1	1	3	3	2	2	2	1	3	3	3	3	2	3	1	1

3 The Evaluation of Testing Methods in AOSE

In this section we evaluate the selected testing methods found in literature for the AOSE approach, according to the framework presented in Section 2. The evaluation and the points that each method has gained, are presented in Table 1.

In the first division of the framework, we evaluate that which testing layers are covered in the method. Then, the test types employed in each test layer are checked, and finally we verify the existence of any metrics and measurements proposed in the method. We rank the MAS test basics of each method (discussed in 2.1), according to the metrics presented in 2.5.

For the purpose of ranking the test process criteria, we emphasize that whenever a method does not explicitly mention the testing activities (discussed in 2.2), we rank it based on a personal analysis that may not necessarily reflect the original intentions of the proposers, and that sometimes has to sharpen shades.

Whether a method performs during the development phase without running the system or not, we simply gave the static testing criterion the ranks of 3 and 1, respectively. We use the same ranking scheme for the dynamic testing criterion, to indicate if a method performs on the running agent-based system.

Based on the tool support, automatic test case generation, visualization and knowledge level that may be suggested by a method, we evaluate the pragmatics aspect of a method (discussed in 2.4). Furthermore, we investigate whether the method proposed particular domain of applicability or development methodology to use the method, or we can use the method within variant methodologies. The only exception here is that the given points indicate the quality of each property in addition to its existence.

Due to lack of space we will not discuss the justification of each given point to a relevant method, which we leave to future work for the selected methods. As illustrated in Table 1, there is no single best method to achieve the highest score in all criteria. Therefore, in order to choose a comprehensive method for testing in AOSE, that includes sufficient essential properties, we have to combine several existing methods.

As it is not possible to join all methods, we need to find the smallest subset of all *interesting* methods. Interesting are all methods that are not worse than any other method in all criteria. We use an approach called *Skyline* [5] to find the subset of interesting methods. Table 2 illustrates the skyline of the evaluated methods, ordered by the total points that each method has gained. Any method left out of this skyline is dominated by at least one method presented in the skyline and can be disregarded in the combination of methods.

The size of the skyline is still large. It shows that most of the proposed testing methods in AOSE, present an approach suitable for at least one group of the MAS developers. Furthermore, the large size of the skyline emphasizes the lack of a comprehensive testing method in AOSE. From the Skyline, we can now make our final decision, thereby weighing our personal preferences for testing criteria.

Table 2 – The Skyline of the Test Methods in AOSE

The Test Method	Total Point
[23] Goal-Oriented	57
[27] Design artifacts	52
[22] Evolutionary Testing	51
[34] SUNIT	50
[28] Test Agent	50
[32,33] Conversation Verification	49
[21] Zeus	48
[15] MadKit	46
[10] SEAUnit	46
[24,25] Ontology-Based	45
[19] Model-based DL detection	44
[13] INGENIAS	43
[29] 3-layer	40

4 Conclusion

In this paper, we investigate the essential aspects that an agent-oriented test method should include. The proposed framework in section 2 divides these aspects into the four major divisions: MASs test basics, test process, test techniques and pragmatism. All these criteria are explained in section 2. Section 3 demonstrates the use of the proposed framework by performing an evaluation of some prominent testing methods. We conclude that there is no method that covers all criteria. As a result, in order to find a sufficient test method we have to combine several different methods. The selected subset of testing methods is presented in Table 2. The combination of these methods fulfills most considered criteria. However, there are some criteria which are disregarded by most of the methods, e.g. the *agent acceptance testing*. Furthermore, the evaluation demonstrates low points in the *performance test*, the *metrics and measurements* and the *test process*. In our future work, we plan to devise a method to promote the testing process in AOSE, by dominating the uncovered criteria and performing a standard testing process, proposed by ISTQB.

5 References

[1] Akbari, Z. "A survey of agent-oriented software engineering paradigm: Towards its industrial acceptance"; Journal of Computer Engineering Research, Vol. 1, Issue 2, pp. 14-28, April 2010.

[2] Ayatollahzadeh Shirazi, M.R., Abdollahzadeh Barfouroush, A. "A Framework for Agent-Oriented

Software Engineering Based On an Analytical Survey"; Iranian Journal of Electrical and Computer engineering, Vol. 6, Issue 1, pp. 36-47, 2007.

[3] Black, R. "Guide to the ISTQB Advanced Certification as an Advanced Test Manager". Rocky Nook, Vol. 1, 2009.

[4] Bordini, R.H., Fisher, M., Visser, W., Wooldridge, M. "Verifying multi-agent programs by model checking"; Autonomous Agents and Multi-Agent Systems, Vol. 12, Issue 2, pp. 239-256, 2006.

[5] Borzsonyi, S., Kossmann, D., and Stocker, K. "The skyline operator"; In Proceedings of the International Conference on Data Engineering (ICDE'01), pp. 421-430, 2001.

[6] Caire, G., Cossentino, M., Negri, A., Poggi, A., and Turci, P. "Multi-agent systems implementation and testing"; From Agent Theory to Agent Implementation - Fourth International Symposium (AT2AI-4), Vienna, Austria, April 2004.

[7] Cernuzzi, L., Cossentino, M., Zambonelli, F. "Process Models for Agent-based Development"; Journal of Engineering Applications of Artificial Intelligence, Vol. 18, Issue 2, pp. 205-222, 2005.

[8] Coelho, R., Cirilo, E., Kulesza, U., von Staa, A., Rashid, A., Lucena, C. "Jat: A test automation framework for multi-agent systems"; In Proceedings 23rd IEEE International Conference on Software Maintenance (ICSM07), pp. 425-434, 2007.

[9] Coelho, R., Kulesza, U., von Staa, A., Lucena, C. "Unit Testing in Multi-Agent Systems using Mock Agents and Aspects"; In Proceedings of the 2006 International Workshop on Software Engineering for Large-Scale Multi-Agent Systems, 83-90, 2006.

[10] Ekinci, E.E., Tiryaki, A.M., Çetin, Ö. "Goal-oriented agent testing revisited"; In proceedings of the Ninth International Workshop on Agent-Oriented Software Engineering, pp. 85-96, 2008.

[11] Gatti, M.A.C., Staa, A.V. "Testing & debugging multi-agent systems: a state of the art report"; Departamento de Informa tica, PUC-Rio, Rio de Janeiro, 2006.

[12] Glossary Working Party. "Standard glossary of terms used in Software Testing"; International Software Testing Qualifications Board, 2010.

[13] Gomez-Sanz, J.J., Botía, J., Serrano, E., Pavón, J. "Testing and debugging of MAS interactions with INGENIAS"; In proceedings of the Ninth International Workshop on Agent-Oriented Software Engineering, pp. 133-144, 2008.

[14] Graham, D., Van Veendendal, E., Evans, I., Black, R.; "Foundations of Software Testing ISTQB Certification". Patrick Bond, 2008.

[15] Huget, M.P., Demazeau, Y. "Evaluating multiagent systems a record/replay approach"; In Proceedings of the IEEE/WIC/ACM International Conference of Intelligent Agent Technology (IAT 2004), pp. 536-539, 2004.

[16] Knublauch, H. "Extreme programming of multi-agent systems"; In Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), pp. 704-711 2002.

[17] Lin, C.E., Kavi, K.M., Sheldon, F.T., Daley, K.M., Abercrombie, K. "A Methodology to Evaluate Agent Oriented Software Engineering Techniques"; In Proceedings of the 40th Hawaii International Conference on System Sciences, pp. 1-10, 2007.

[18] Low, C.K., Chen, T.Y., Rönnquist, R. "Automated Test Case Generation for BDI agents"; Autonomous Agents and Multi-Agent Systems, Vol. 2, Issue 4, pp. 311-332, 1999.

[19] Mani, N., Garousi, V., Far, B.H. "Testing Multi-Agent Systems for Deadlock Detection Based on UML Models"; In proceedings of the 14th International Conference on Distributed Multimedia Systems (DMS08), Boston, USA, pp. 77-84, 2008.

[20] Moreno, M., Pavon, J., Rosete, A. "Testing in Agent Oriented Methodologies"; Omatu et al. (Eds.): IWANN, Part II, LNCS 5518, Springer-Verlag Berlin Heidelberg, pp. 138-145, 2009.

[21] Ndumu, D.T., Nwana, H.S., Lee, L.C., Collis, J.C. "Visualization and debugging distributed multi-agent systems"; In Proceedings of the third annual conference on Autonomous Agents (ACM press), pp. 326-333, 1999.

[22] Nguyen, C.D., Perini, A., Tonella, P. "Constraint-based Evolutionary Testing of Autonomous Distributed Systems"; In proceedings of IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08), pp. 221-230 2008.

[23] Nguyen, C.D., Perini, A., Tonella, P. "A goal-oriented software testing methodology"; Luck, M., Padgham, L. (eds.) Agent-Oriented Software Engineering VIII. LNCS, Springer, Heidelberg, vol. 4951, pp. 58-72, 2008.

[24] Nguyen, C.D., Perini, A., Tonella, P. "Experimental Evaluation of Ontology-Based Test Generation for Multi-agent Systems"; M. Luck J.J., Gomez-Sanz (eds.): Agent-Oriented Software Engineering, Springer-Verlag Berlin Heidelberg, pp. 187-198, 2009.

[25] Nguyen, C.D., Perini, A., Tonella, P. "Ontology-based test generation for multi agent systems (short paper)"; In Proceedings of the 7th International

Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008). Estoril, Portugal, pp. 12-16, 2008.

[26] Padgham, L., Winikoff, M., Poutakidis, D. "Adding Debugging Support to the Prometheus Methodology"; Engineering Applications of Artificial Intelligence, special issue on Agent-oriented Software Development, Volume 18, Issue 2, pp. 173-190, March 2005.

[27] Poutakidis, D., Winikoff, M., Padgham, L., Zhang, Z. "Debugging and Testing of Multi-Agent Systems using Design Artefacts"; R.H. Bordini et al. (eds.), Multi-Agent Programming, Springer Science + Business Media, pp. 215-258, 2009.

[28] Rouff, C. "A Test Agent for Testing Agents and Their Communities"; Aerospace Conference Proceedings, IEEE Volume 5, pp. 2633-2638, 2002.

[29] Salamon, T. "A Three-Layer Approach to Testing of Multi-agent Systems"; G.A. Papadopoulos et al. (eds.): Information Systems Development, DOI 10.1007/b137171_41, Springer ScienceBusiness Media, pp. 393-401, 2009.

[30] Srivastava, P., R., Anand, K., Reddy, S., Raghurama G. "Regression Testing Techniques for Agent Oriented Software"; In Proceedings of the International Conference on Information Technology, pp.221-225, 2008.

[31] The V-Model: The Development Standards for IT Systems of the Federal Republic of Germany 2005, *http://www.v-modell.iabg.de/* (cited February 2011).

[32] Timothy, H.L., DeLoach, S.A. "Automatic Verification of Multiagent Conversations"; In the Annual Midwest Artificial Intelligence and Cognitive Science Fayetteville, 2000.

[33] Timothy, H.L., DeLoach, S.A. "Verification of Agent Behavioral Models"; In Proceedings of the International Conference on Artificial Intelligence (IC-AI'2000), 2000.

[34] Tiryaki, A.M., Oztuna, S., Dikenelli, O., Erdur, R.C. "Sunit: A unit testing framework for test driven development of multi-agent systems"; Padgham, L., Zambonelli, F. (eds.): AOSE VII / AOSE 2006. LNCS, Springer, Heidelberg, vol. 4405, pp. 156-173 2007.

[35] Sturm A., Shehory, O. "A Framework for evaluating agent-oriented methodologies"; In Proceedings of Workshop on Agent-Oriented Information System (AOIS), Melbourne, Australia, pp. 60-67, 2003.

[36] Winikoff, M. "Future Directions for Agent-Based Software Engineering"; Special section on Future of software engineering and multi-agent systems, International Journal of Agent-Oriented Software Engineering (IJAOSE08), pp. 1-10, 2008.

Modeling of Object Oriented Software Testing Cost

Dinesh Kumar Saini[1], Moinuddin Ahmad[2]
1Faculty of Computing and Information Technology, Sohar University
2Faculty of Business , Sohar University
P.O. Box: 44, P.C. 311, Sohar, Sultanate of Oman
Tel: +968-26720101 Ext: 251, Mobile: +968-95784762, +968-95784715
e_mail: dinesh@soharuni.edu.om, mahmad@soharuni.edu.om

Abstract *An effort is made to develop a cost model in this paper for the object oriented software testing process. The software must be tested to an extent so that before releasing it. is assured that the failures risks have been minimized. There is a trade-off between testing cost and release policies. There may be various factors affecting the total software testing cost. In this paper, an object oriented software testing cost model is presented which can be used to determine the optimal release policies for the software. Various software cost factors are considered in this paper such as the cost of object oriented testing and removing detected errors. The error which incurs lot of money and time in the software development process must be included as removal cost which is a stochastic process*.

Key Words Object Oriented Systems, Testing, Cost, Error, and Bug.

1. Introduction

Once the software developing process (usually including the following four phases: analysis, design, coding, and testing) is completed, the software company releases the software product to market and obtains some profits in return if the software functions successfully (Mili, A. et al. 2000). During the developing process, the company has to pay for such expenditures as the labor fee, the cost of developing, and the cost of testing, and so on. Therefore there is always testing cost associated whenever we want to make quality software. Customer satisfaction depends on the quality of the software, higher the quality higher the customer satisfaction but finally the customer has to pay for all the expenditures occurred during the software testing. Therefore it becomes the moral duty of software developing company to justify the software testing cost in terms of software quality achieved.

Therefore it is important to have an estimation of software testing cost in some mathematical expression. In the software development process, the manager must reasonably decide when to stop testing and release the software system to the user. This decision problem is called optimal software-release (Ohtera H. et al. 1990). The pressure of delivering high quality software products calls for a better understanding of the software development process and improved software models (Koch H.S. et al.1983). The cost of software testing can be high, and in many cases it is exceeding 30% of the overall software development costs for the project (Hetzel et al. 1988). This places increased pressure on testing team. The software products which are developed using object oriented concepts needs a decision as when to stop testing based on some criteria. We develop a mathematical expression for all testing costs that may occur during testing and removing errors in object oriented development.

In order to improve the quality of software products, testing serves as the main tool to remove faults in software products. There will be exponential increase in the cost when quality enforcement is carried out at the refinement stage in the software development process refinement (Pham H. et al. 1999). Therefore, it is usually very important to determine when to stop testing based on the cost assessment.

2. Object Oriented Software testing Activities

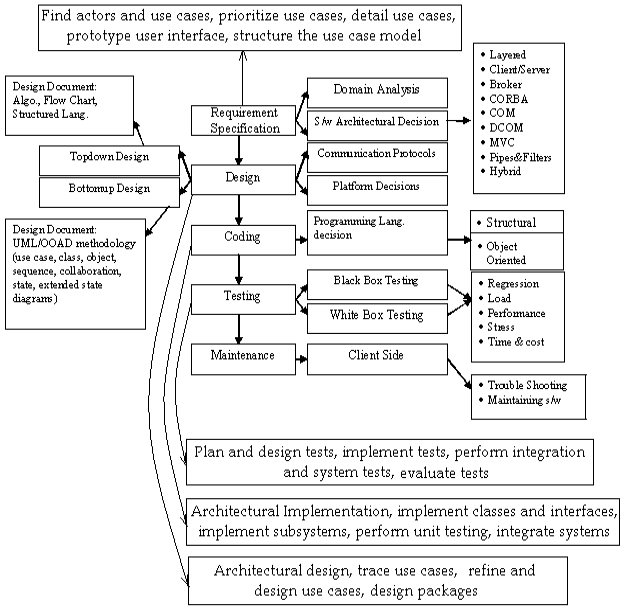


Figure 1 the various testing activities in SDLC

In object oriented software development life cycle the various phases are shown in Figure 1. The various activities associated with these phases are also shown here. Activities are the things that are expected from the designers and what they should be doing. An activity takes inputs and produces outputs. These inputs and outputs are referred to as artifacts. The artifacts that act as an input to a particular activity could be a use case, while the output from that activity could be a class diagram etc.

3. Software testing cost

Software systems are prone to failures and the failure can occurs any time or even it can be at any random times. Failures are caused by the faults in the systems.

Software testing is very critical part of the software because it helps in reducing the chances of failures. The cost of software development process has big portion for the software testing. Accurately estimating the costs of all phases of the software development process as well as software testing have become more important. Since resources are limited, it is critical to determine how they should be allocated throughout the software life-cycle (Briand L.C. et al. 2002). Various tools are also being developed to estimate software testing costs.

Since the main objective of testing is to improve the software quality. Neither process improvements in development nor in testing alone can guarantee quality improvement. Harter and Slaughter (Harter, D. E. et al. 2000) state that it is an unresolved issue how software quality can be improved.

Quality can be tested into software products or it can be designed or built into the products (Slaughter, S. A., et al. 1998). According to Harter and Slaughter (Harter, D. E. et al. 2000), software quality is rather designed than tested into products. Both design and testing have a significant influence on product quality.

The software cost estimation should be done correctly. Some framework like (Grimstad, S. et al. 2006) can be used for correctly estimating the cost.Here we will try to model the object oriented testing cost. Cost estimation (Dolado J.J., 2001) can be erroneous and some strategy like (Lederer A.L. et al. 1988) can be used to estimate the cost estimation errors. Various researchers have also used experiments to predict the project costs using different algorithmic techniques (Banker, R. D. et al. 1989), (Kemerer, C. F. 1987), (Kusters, R. et al. 1990), (McCabe; 1976). These can also be used for estimating the cost of testing the software.

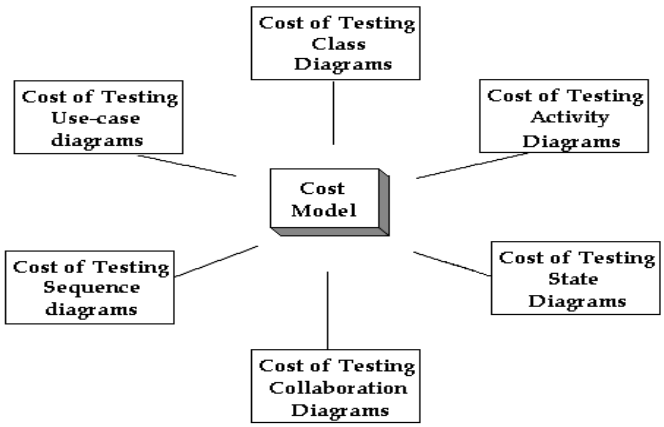


Figure 2. The Cost Diagram for Testing of Object Oriented Systems

4. Cost Model

Object-oriented testing needs to test all the aspects of software development process. An object oriented software development consists of various phases in its development cycle. Each phase of software development needs to be tested for correctness. Object oriented softwares are developed using UML specifications. These UML diagrams (Booch, G. et al. 1998) can themselves be used to automatically generate and select test cases (Kansomkeat, S. et al. 2003) (Offutt, J. et al.1999). A use case is a set of scenarios that describes an interaction between a user and a system. Use case diagrams are related and this relationship is visible among actors and use cases. The two main components of a use case diagram are use cases and actors. There may be various steps in which use cases may be tested (Booth, G., 1986).

First of all the use cases are required to be tested syntactically to ensure that the use case descriptions describes the correct and proper information. We must find answers of questions like: Is it complete? Is it correct? Is it consistent? After checking for the syntax next step is domain testing. Here we need to find a domain expert. Again, we need to ask the same questions: Is it complete? Is it correct? Is it consistent?

After that the next step is traceability testing. Traceability testing is to make sure all the requirements especially functional requirement are there in the use cases and from the use cases to the requirements back. It should be complete, correct and consistent and this should be tested.

Let the number of use cases is N_u and any ith use case takes T_i time in testing and the cost associated with testing one use-case testing be K_u then total cost of use case testing is given by:

$$C_{use\ case} = \sum_{i=1}^{N_u} K_u T_i \qquad \text{----------(1)}$$

Class diagrams in object oriented systems are description of the types of objects in object oriented systems and their relationships with each other. Class diagrams helps in modeling the class contents and class structures. Class

Diagrams uses design tools and elements of the class. Class diagrams express three different aspects of the systems like conceptual, specification, and implementation. Classes consist of three main items which are: a name, attributes, and operations. All the three are needed for testing the class concept and the cost of testing a class diagram will depends on the following items.

1. The number of classes in a class diagram let it be N_C.
2. The number of relationships of each of the class with other classes, let any class C_i is having relationships with n_i other classes.
3. The number of attributes and the number of operations that each class has, let a class C_i has total number of attributes A_i and total number of operations O_i.

So the cost of testing a class diagram will be:

$$C_{\text{Class Diagram}}=N_C \sum_{i=1}^{N_C} K_C n_i \quad \text{----------(2)}$$

The other major component of object oriented systems which should be tested seriously is interaction diagrams, which are responsible for modeling the behavior of use cases by describing the way groups of objects interact to complete the task in the object oriented systems. The two major kinds of interaction diagrams are sequence diagrams and collaboration diagrams. Interaction diagrams are helpful in describing the behavior of objects in the use cases. Interaction diagrams provide information for how the objects collaborate behaves in the object oriented systems. . Sequence diagrams demonstrate the behavior of objects in a use case by describing the objects and the messages they pass. There may be a number of sequence diagrams. Let the number of sequence diagrams to test is N_S and the cost associated with testing one sequence diagram is K_S so the total cost of testing sequence diagrams is given as:

$$C_{\text{Sequence Diagram}}= \sum_{i=1}^{N_S} K_S n_i \quad \text{----------(3)}$$

Collaboration diagrams describes the relationship between objects and the order of messages passed between objects. The objects are listed as icons and arrows indicate the messages being passed between them. The numbers next to the messages are called sequence numbers. As the name suggests, they show the sequence of the messages as they are passed between the objects. Let the number of collaboration diagrams is N_{Cl} and the cost associated with one collaboration diagram is K_{Cl} then the total cost of testing collaboration diagrams is:

$$C_{\text{Collaboration Diagram}}= \sum_{i=1}^{K_{Cl}} K_{Cl} n_i \quad \text{----------(4)}$$

State diagrams are used to describe the behavior of a system. State diagrams describe all of the possible states of an object as events occur. Each diagram usually represents objects of a single class and tracks the different states of its objects through the system. The state diagram testing cost will depend on the number of state diagrams N_{St} and the number of states in each state diagram, let number of states in ith state diagram is n_i.

$$C_{\text{State Diagram}}= \sum_{i=1}^{N_{St}} K_{St} n_i \quad \text{----------(5)}$$

Activity diagrams describe the workflow behavior of a system. Activity diagrams are similar to state diagrams because activities are the state of doing something. The diagrams describe the state of activities by showing the sequence of activities performed. Activity diagrams can show activities that are conditional or parallel. The cost of testing activity diagrams will depend on number of activity diagrams to be tested in program and the number of activities in each of the activity diagrams. Let there be total N_A activity diagrams ate there and number of activities in ith activity diagram is n_i

$$C_{\text{Activity Diagram}}= \sum_{i=1}^{N_A} K_A n_i \quad \text{----------(6)}$$

Software testing is done at various levels, so the different software test cost is associated with these various levels. These are basically unit testing, integration testing and system testing. Unit testing is concerned with verifying the behavior of the smallest isolated components of the system. Typically, this kind of testing is performed by developers or maintainers and involves using knowledge of the code itself. In practice, it is often difficult to test components in isolation. Components often tend to rely on others to perform their function. In object oriented environment the unit is a class.

Integration testing is focused at the verification of the interactions between the components of the system. The components are typically subjected to unit testing before integration testing starts. A strategy that determines the order in which components should be combined usually follows from the architecture of the system.

System testing occurs at the level of the system as a whole. On the one hand, the system can be validated against the non-functional requirements, such as performance, security, reliability or interactions with external systems. On the other hand, the functionality implemented by the system can be compared to its specification (Farren D, 1996).
Once the software developing process (usually including the following four phases: analysis, design, coding, and testing) is completed, the software company releases the software product to market and obtains some profits in return if the software functions successfully. The product may not be good enough if all the metrics involved in software test are not well tested (Kan, S.H. et al. 2001).

5. Testing cost model for object oriented systems

How the cost of object oriented software varies? What are the factors on which it depends on? Our proposed model tries to find out the answer of these questions. There are various other models (Yang, M.C. et al. 1995) (Goel A.L. et al.1979] in literature which model the optimum release time for the software. There may be software release policies determined by company management for this. So this estimation of software test cost for object oriented environment is a critical issue (Page T., et al, 1989). Our model of software testing broadly divides the whole testing process as according to its testing levels.

We have taken into consideration the factors into consideration which affect in each of these phases. Our model consists of:

1. Cost to perform testing E_1 (T).
2. Cost incurred in removing errors E_2 (T).

The Models assumptions are as follow:

- The cost to perform testing is proportional to the testing times.
- Errors found during testing consist of two parts - the deterministic part and the incremental random part of the error.
- The testing of a class consists of – testing all of the class components and its association with other classes.

Notation:

C_1: Software unit testing cost per unit time
C_2: Software integration testing cost per unit time
C_3: Software system testing cost per unit time
C_4: deterministic cost to remove each error per unit time during unit testing phase
C_5: deterministic cost to remove each error per unit time during integration testing phase
C_6: deterministic cost to remove each error per unit time during system testing phase
T_U: Time incurred in unit testing
T_I: Time incurred in integration testing
T_S: Time incurred in system testing
w_U: random variable of cost to remove errors during unit testing
w_I: random variable of cost to remove errors during integration testing
w_S: random variable of cost to remove errors during system testing
M: Total number of classes
C_{w_U} : expectation value of variable w in unit testing
C_{w_I} : expectation value of variable w in integration testing
$m(T_U)$: average number of failures in all classes detected during the unit testing time T_U

a. Testing Cost:

Includes testing activities: preparing test cases, running them and analyzing results.

$E_1(T)=C_1T_U + C_2T_I + C_3T_S$ ----------(7)
where $T_U + T_I + T_S = T$

b. Error removal cost:

Errors are removed as soon as they are discovered. We will find out the expressions of error removal cost at various levels of testing. These levels are unit testing, integration testing and system level testing.

If we consider the testing at unit level, then here the unit is class. Let us consider that there are M numbers of classes which are tested for errors. Testing of a class means we have to test each of its methods.
According to assumption (2) above, the cost to remove errors during the testing phase is a random variable and consists of two parts - the deterministic part, say C_4, and the incremental random part, which reflects the difficulties of different errors.
(N(t), t>0) is counting process of errors detected during testing phase and can be considered a stochastic renewal process.
The cost of removing j'th error in a class:
$Cj = C_4 + W$
The cost of removing all errors from class any class *i* is given by:

$$\sum_{j=1}^{N_i(T_i)} C_4 + w \quad \text{----------} \quad (8)$$

Where ($N_i(t_i>0)$ is the counting process of errors detected during testing of class L_i . So $N_i(T_i)$ is the total number of errors discovered in time T_i .
Expected total cost to remove all detected errors in a class during period $[0,T_i)$, $E_{2i}(T_i)$ is given by:

$$E_{2i}(T_i) = E\left[\sum_{j=1}^{N_i(T_i)} C_4 + w_U\right] \text{----------(9)}$$

If there are total M classes, then cost of removing all errors from M classes:

$$E_U(T_U) = \sum_{i=1}^{M} E\left[\sum_{j=1}^{N_i(T_i)} C_4 + w_U\right]$$

$$= \sum_{i=1}^{M}\sum_{n_i=1}^{\infty}\left\{E\left[\sum_{j=1}^{N_i(T_i)} C_4 + w_U\right].P(N_i(T_i)=n_i)\right\}$$

$$= \sum_{i=1}^{M}\sum_{n_i=1}^{\infty}\left\{E\left[\sum_{j=1}^{n_i} C_4 + w_U\right].P(N_i(T_i)=n_i)\right\}$$

$$= \sum_{i=1}^{M} \sum_{n_i=1}^{\infty} \{[n_i C_4 + C_{w_U}] P(N_i(T_i) = n_i)\}$$

$$= \sum_{i=1}^{M} \left\{ C_4 \sum_{n_i=1}^{\infty} n_i . P(N_i(T_i) = n_i) + C_{w_U} \sum_{n_i=1}^{\infty} P(N_i(T_i) = n_i) \right\}$$

$$= \sum_{i=1}^{M} \{C_4 E(N_i(T_i)) + C_{w_U}\}$$

$$= \sum_{i=1}^{M} \{C_4 m_i(T_i) + C_{w_U}\}$$

$$= C_4 \sum_{i=1}^{M} m_i(T_i) + C_{w_U} M$$

$$= M[C_4 m(T_U) + C_{w_U}] \text{ where}$$

$T_U = T_1 \cup T_2 \cup T_3 \cup ..... \cup T_M$

and $m(T_U)$ is average number of failures in all classes detected during the unit testing time T_U.

The cost incurred in error removal during integration testing:

the cost of removing i'th error

$C_i = C_5 + w_I$

$$E_I(T_I) = E\left[\sum_{i=1}^{N(T_I)} C_i\right]$$

$$= E\left[\sum_{i=1}^{N(T_I)} (C_5 + w_I\right]$$

$$= \sum_{n=1}^{\infty} \left\{ E\left[\sum_{i=1}^{N(T_I)} (C_5 + w_I\right] . P(N(T) = n) \right\}$$

$$= \sum_{n=1}^{\infty} \left\{ E\left[\sum_{i=1}^{n} (C_5 + w_I\right] . P(N(T) = n) \right\}$$

$$= \sum_{n=1}^{\infty} \{[nC_5 + C_{w_I}] P(N(T) = n)\}$$

$$= C_5 \sum_{n=1}^{\infty} n . P(N(T) = n) + \sum_{n=1}^{\infty} C_{w_I} P(N(T) = n)$$

$$= C_5 E(N(T)) + C_{w_I}$$

$$= C_5 m_I(T_I) + C_{w_I}$$

Similar expression is obtained for system level testing:

$E_S(T_S) = C_6 m_S(T_S) + C_w$

so the total cost to remove errors

$$= \sum_{i=1}^{N_u} K_u T_i + \mathrm{N_C} \sum_{i=1}^{N_C} K_C n_i + \sum_{i=1}^{N_S} K_S n_i + \sum_{i=1}^{K_{Cl}} K_{Cl} n_i +$$

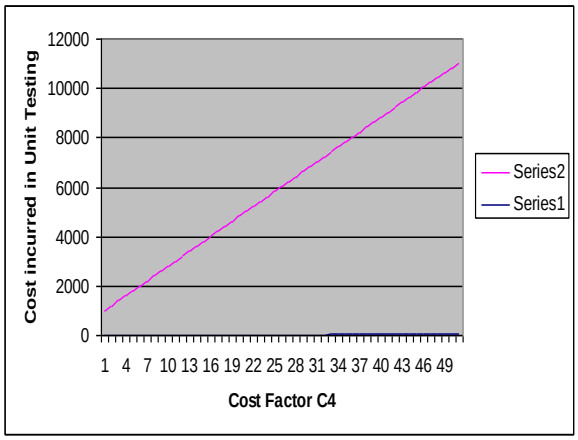
$$\sum_{i=1}^{N_{St}} K_{St} n_i + \sum_{i=1}^{N_A} K_A m_i +$$

$$M[C_4 m(T_U) + C_{w_U}] + C_5 m_I(T_I) + C_{w_I} + C_6 m_S(T_S) + C_{w_S}$$

c. Impact of Testing cost coefficient C_4

In unit test the cost of removing all errors from a class depends on cost factor C_4 and the random factor W. From the expression obtained for total cost incurred in unit test it is clear that total cost is linear with C_4. To plot the graph between total unit test cost and C_4 We have taken M=20, $m(T_U) = 10$ and $C_{w_U} = 50$

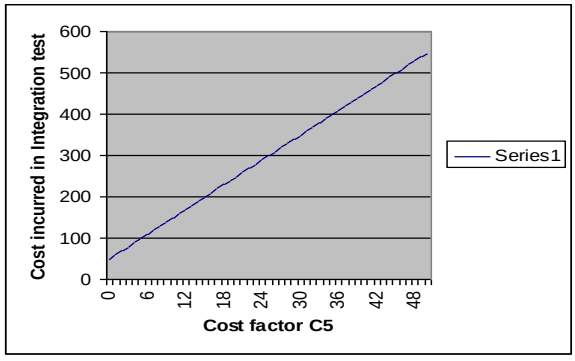
Its clear that for more number of classes the cost would be increased with that factor. It is evident that the cost of testing is increasing with the testing time and number of faults appearing in the software.



(SOURCE: Wipro Software Testing Unit)

d. Impact of Testing cost coefficient C_5

The cost incurred in integration testing depends on cost factor C_5. From the expression obtained for total integration testing cost it is observed that it is linear with C_5.



(SOURCE: Wipro Software Testing Unit)

6. Conclusion

The testing cost in object oriented software system has a great impact on deciding the release time for the software. The more we test the cost incurred will also increase more. But the overall completeness of software will be significantly increased. Therefore there must be a trade-off between testing cost and time. Our proposed model can be used for predicting the total cost incurred in testing and removing errors and bugs from object oriented systems.

References:

1. Ohtera H., Yamada S, 1990, "Optimum Software-Release Time Considering an Error-Detection Phenomenon during Operation", IEEE transactions on reliability. Vol. 39, no. 5. pp.596-599.
2. Koch H.S., Kubai P., 1983, "Optimal release lime of computer software".IEEE Trans. Software Engineering, vol SE-9, pp 323-327.
3. Yang, M.C. and Chao A., 1995, "Reliability-Estimation & Stopping-Rules for Software Testing, Based on Repeated Appearances of Bugs," IEEE Transaction on Reliability 44, 2, pp. 315-326.
4. Goel A.L.and Okumoto K, 1979, Time-Dependent Error-Detection Rate Model for Software Reliability and Other Performance Measures," IEEE Ttans. Reliability, Vol. R-28, No. 3, pp.206-211.
5. Page T., et al, 1989, "An Object Oriented Modelling Environment," Proceedings of the Fourth Annual ACM Conference on Object Oriented Programming Systems, Languages and Applications (OOPSLA).
6. Farren D, 1996, The Economics of System Level Testing, PhD thesis, Brunei Univ., Uxbridge, U.K.
7. Hetzel, William C., 1988, The Complete Guide to Software Testing, Second edition, John Wiley & Sons, NY USA.
8. Pham H. and Zhang X., 1999, "Software release policies with gain in reliability justifying the costs", .Annals of Software Engineering 8, pp.147-166.
9. Mili, A., Chmiel, S. F., 2000, Gottumukkala, R., and Zhang, L. 2000. An integrated cost model for software reuse. In Proceedings of the 22nd international Conference on Software Engineering (Limerick, Ireland, June 04 - 11, 2000). ICSE '00. ACM Press, New York, NY, pp.157-166.
10. Booth, G., 1986, "Object Oriented Development," IEEE Transactions on Software Engineering, SE-12, February, pp. 211-221.
11. Kan, S.H., Parrish, J., and Manlove, D., 2001, In-process metrics for software testing, IBM Systems Journal, Vol. 40, No. 1, 2001, p. 220.
12. Harter, D. E. and Slaughter, S. A. 2000. Process maturity and software quality: a field study. In *Proceedings of the Twenty First international Conference on information Systems* (Brisbane, Queensland, Australia). International Conference on Information Systems. Association for Information Systems, Atlanta, GA, pp. 407-411.
13. Kansomkeat, S. and Rivepiboon, W. 2003. Automated-generating test case using UML statechart diagrams. ACM International Conference Proceeding Series, vol. 47. South African Institute for Computer Scientists and Information Technologists, pp. 296-300.
14. Booch, G., Rumbaugh, J., and Jacobson, I. 1998. The Unified Modeling Language User Guide. Object Technology Series. Addison Wessley Longman, Inc.
15. Offutt, J., and Abdurazik, A. 1999. Generating test cases from UML specifications. In Proceeding of the 2nd International Conference on the Unified Modeling Language (UML99) , Fort Collins, CO, October, 1999.
16. Slaughter, S. A., Harter, D. E., and Krishnan, M. S. 1998, Evaluating the Cost of Software Quality. Communications of the ACM, 41, 8 (1998), pp. 67-73.
17. Grimstad, S. and Jørgensen, M. 2006. A framework for the analysis of software cost estimation accuracy. In Proceedings of the 2006 ACM/IEEE international Symposium on international Symposium on Empirical Software Engineering (Rio de Janeiro, Brazil, September 21 - 22, 2006). ISESE '06. ACM Press, New York, NY, pp. 58-65.
18. Briand L.C. and Wieczorek I., 2002, "Resource estimation in software engineering," in Encyclopedia of software engineering, J. J. Marcinak, Ed., 2nd ed. New York: John Wiley & Sons, pp. 1160-1196.
19. Lederer A.L. and Prasad, J. 1988 "A causal model for software cost estimating error," IEEE Transactions on Software Engineering, vol. 24, no. 2, pp. 137-148.
20. Dolado J.J., 2001, "On the problem of the software cost function," Information and Software Technology, vol. 43, no. 1, pp. 61-72.
21. Martin, R. 1988 "Evaluation of Current Software Costing Tools," Software Eng. Notes, vol.13, no.3, pp.49-51.
22. Banker, R. D. and Kemerer, C. F. 1989. Scale Economies in New Software Development.

IEEE Trans. Softw. Eng. 15, 10 (Oct. 1989), pp.1199-1205.

23. Kemerer, C. F. 1987. An empirical validation of software cost estimation models. Commun. ACM 30, 5 (May. 1987), pp.416-429.
24. Kusters, R., van Genuchten, M. J., and Heemstra, F. J. 1990. Are software cost-estimation models accurate. Inf. Softw. Technol. 32, 3 (Apr. 1990), pp.187-190.
25. McCabe; 1976, A Complexity Measure, IEEE Transactions on Software Engineering, 2, pp.308-320.

Software Defect Taxonomy, Analysis and Overview

Ali A Karahroudy, M.H.N. Tabrizi
Asgharykarahroudy10@students.ecu.edu, Tabrizim@ecu.edu
Computer Science Department, East Carolina University, Greenville, NC, USA

Abstract - *In this paper an overall analysis of current defect taxonomies is presented also plans for well defined process based taxonomy is carefully created using the existing models. The existing software defect taxonomies do not focus fully on the process, in most cases process and product are studied in parallel and significant amount of time is spent to identify and debug the defects or prevent them from occurring again. Our study is focused on the defects found based on the process in which they are found and selected based on the largest potential impact on the final product that will have significant impact in defect prevention.*

Keywords: Defect, Taxonomy, Process, Model, Frame ware, Attribute

1. Introduction

Various studies attempt to quantify the losses caused by software defects. As a recent instance Toyota has announced a recall for its 2010 hybrid vehicles. A break problem caused by faulty antilock braking software is believed to be the reason [1]. Tens of thousands of medical devices were recalled in March 2007 to correct a software bug [2]. Another example is the first flight of the European Space Agency's new Ariane 5 rocket failed shortly after launching [3], resulting in an estimated uninsured loss of a half billion dollars. It was reportedly due to the lack of exception handling of a floating-point error in a conversion from a 64-bit integer to a 16-bit signed integer.

It is also reported that [4] software bugs caused the bank accounts of 823 customers of a major U.S. bank to be credited with $924,844,208.32 each in May 1996, according to newspaper reports. The American Bankers Association claimed it was the largest such error in banking history. According to bank spokesman the programming errors were corrected and all funds were recovered. One study [5] estimated losses to the U.S. economy alone due to software defects at $60 billion annually.

There are many instances of monumental software failures that have staggering losses up to and including the loss of human life. In 2010, a software problem caused bank cards to fail across Germany [6]**.** It is also likely that Air France flight 447 was brought down due to software defect that was not able to handle the extreme weather conditions. In 1999, the Mars Climate Orbiter was lost due to software confusing pounds with kilograms. One of the most notable instances of software failure was the Therac-25 radiation treatment system [7]. In this case, faulty software caused patients to be given massive overdoses of radiation, killing several of them.

Defect prevention in early stages of software development life cycle is very cost effective. Meanwhile the more defect prevention processes are in place, the more costs will be imposed to the project budget therefore a very well balanced model is needed to fulfill both defect prevention and cost effectiveness.

However Software's complexity and accelerated development schedules make defects prevention difficult. Also current models like peer reviews, analysis tools, and different testing techniques detect different classes of defects at different points in the development cycle. [8] providing the justification that no existing model can be enough as a standalone model.

Organizations are still asking how they can predict the quality of their software despite the substantial research effort over the last 30 years. [9] where wide range of prediction models have been proposed. Complexity and size metrics have been used in an attempt to predict the number of defects a system will reveal in operation or testing. Reliability models have been developed to predict failure rates based on the expected operational usage profile of the system. The maturity of design and testing processes has been

advanced as ways of reducing defects. Recently large complex multivariate statistical models have been proposed in an attempt to find a single complexity metric that will account for defects [9].

1. Software Defect Taxonomy

It is reported that the best way to prevent and control software defects is using proper defect taxonomy [10] (A defect is a structural property of software document of any kind, namely a deviation from the nearest correct document that makes the document incorrect or locally incorrect. Taxonomy is a system of hierarchical categories designed to be useful aid for reproducibly classifying things) the area of software quality measurements and quantification is beset with undue complexity and has, in some ways, advanced away from the developer [11]. In an area where the processes are so amorphous, the tangibles required for measurement and modeling are few. With the result academic pursuits that can't be confined to the limitations of practice evolved and became distanced from the developer. In this area, the need to derive tractable measurements that are reasonable to undertake and intuitively plausible cannot be understated. Measurement without an underlying theme can leave the experimentalist, the theorist and the practitioner very confused.

Defect removal and defect prevention techniques [12] are no longer good enough to inspire confidence for software products. Techniques that help predict the number of remaining defects in software products can further boost customer confidence. Such techniques are easy to perform and have been used outside the realm of software engineering to produce reliable estimates for decades in the area of animal, bird, fish, and insect counts, and more recently for estimating the prevalence of "SARS" (Severe Acute Respiratory Syndrome) and cancer occurrences.

In this paper we will review major taxonomies that are being used by software developers.

1.1. Orthogonal Defect Classification

The ODC (Orthogonal defect classification) [13] is a scheme to capture the semantics of each software defect quickly. It is the definition and capture of defect attributes that make mathematical analysis and modeling possible. Analysis of ODC data provides a valuable diagnostics method for evaluating the various phases of the software life cycle (design, development, test and service) and the maturity of the product. ODC makes it possible to push the understanding and use of defects well beyond its quality.

An evolved model based on ODC is introduced by Madachy and Bohme [14] is called Orthogonal Defect Classification COnstructive QUALity Model (ODC COQUALMO) that predicts defects introduced and removed, classified by ODC types. Using parametric cost and defect removal inputs, static and dynamic versions of the model help one determine the impacts of quality strategies on defect profiles, cost and risk.

The dynamic version provides insight into time trends and is suitable for continuous usage on a project. The models are calibrated with empirical data on defect distributions, introduction and removal rates; and supplemented with Delphi results for detailed ODC defect detection efficiencies. This work has supported the development of software risk advisory tools for NASA flight projects. They have demonstrated the integration of ODC COQUALMO with automated risk minimization methods to design higher value quality processes, in shorter time and with fewer resources, to meet stringent quality goals on projects. There are different implementations of ODC COQUALMO as static versions in a spreadsheet and one that runs on the Internet that estimate the final levels of defects for the ODC categories. They have developed a dynamic tool to apply COQUALMO in the field that could be found at [15].Different methods for risk analysis and reduction have been performed in conjunction with ODC COQUALMO, which can produce optimal results in less time

Another technique to reduce risks with the model is a strategic method of optimization. It generates optimal risk reduction strategies for defect removal for a given budget, and also computes the best order of activities. An integration of ODC COQUALMO has also been prototyped with the DDP risk management tool which uses fault trees to represent the overall system's dependencies on software functionality. These experiments to optimize quality processes are described in more detail in [16].

1.2. Defect Severity and Defect Priority

Based on [17] the severity framework for assigning defect criticality that has proven that a five level criticality scale is the most effective scale to study defects. The criticality associated with each level is based on the answers to several questions:

- It must be determined if the defect resulted in a system failure.
- The probability of failure recovery must be determined.
- It must be determined if the system can do this on its own or if remedial measures must be implemented in order to return the system to reliable operation.
- It must be determined if the system can operate reliably with the defect present if it is not manifested as a failure.
- It must be determined if the defect should or should not be repaired.

The five level scale of defect criticality addresses the above mentioned questions are; critical, major, average, minor and exception. In addition to the defect severity level defined above, defect priority level can be used with severity categories to determine the immediacy of repair. A five repair priority scale has also been used in common testing practice. The levels are: resolve immediately, give high attention, normal queue, low priority, and defer.

1.3. Statistical Defect Models

The goal of statistical defect modeling, which includes what is commonly referred to as *Software Reliability Growth* [18], has been to predict the reliability of a software product. Typically, this may be measured in terms of the number of defects remaining in the field, the failure rate of the product, the short term defect detection rate, etc. Although this may provide a good report card, it often occurs so late in the development cycle that makes it of little value to the developer. Ideally, a developer would like to get feedback during the development life cycle.

1.4. Qualitative Casual Analysis

To identify the root causes of the defects, the defects are analyzed, one at a time, by a team that is knowledgeable in the area.

At IBM, the Defect Prevention Process and similar efforts elsewhere have found causal analysis to be very effective in reducing the number of errors committed in a software project [19]. The qualitative analysis provides feedback to developers that eventually improve both the quality and the productivity of the software organization. Defect prevention can provide feedback to developers at any stage of their software development process.

However, the resources required to administer this method are significant, although the rewards have proven to be valuable. Moreover, given the qualitative nature of the analysis, the method does not lend itself well to measurement and quantitative analysis. Consequently, defect prevention, though not a part of the engineering process control model, could eventually work in conjunction with it.

1.5. Peer Review Technique

Peer reviews, in particular software inspections, have become accepted within the software industry as a cost effective way of removing defects as early as possible in the software development cycle [20] The peer review process is also quite easy to measure.

Peer reviews are included in the Software Engineering Institute's Capability Maturity Model Integration (CMMI ®) [21] as a required process for those organizations following the CMMI as a guide for process improvement. Peer reviews are especially valuable as a way to remove defects early in the development cycle and should be performed on all major software development work products including requirements, design, code, and test procedures. The software inspection method of performing peer reviews is generally considered the most effective method of performing peer reviews [22] and is an indisputable software engineering best practice. Other types of peer reviews that are practiced with varying degrees of formality are team reviews, walkthroughs, and pair programming. Once peer reviews are an established practice, the data from each peer review can be used for defect management. For this purpose the following data from each peer review are recommended to be collected [23]:

- Program, function, and work product identifiers
- Type and phase of review, e.g., design walkthrough or code inspection
- Who attended and how much time was spent preparing for the review meeting
- How long the review meeting(s) lasted and who attended the meeting
- Size of the work product, e.g., pages of design or source lines of code (SLOC)
- Total major defects and total minor defects detected

For each defect found the following data is recommended [24]:

- Defect type, e.g., missing, wrong, or extra.
- Defect origin, i.e., the development phase where the defect originated, e.g., requirements, design or code. Note that it is possible that a defect can be discovered in a later phase than when it was first inserted, e.g., a design defect can be discovered during a peer review of code.
- Defect severity, i.e., major or minor. A major defect being any defect that could be discovered or observed by a customer during operational use of the software, or a defect that requires significant time to fix. Minor defects are everything else, e.g., documentation errors.
- Defect location, e.g., module or program element name.

And finally from the collected data the following derived measurements are recommended for each peer review:

- (Major) defects per detection hour – the average number of major defects found for each hour of detection time derived by dividing the number of major defects found by the sum of the participants' total preparation time and total time (labor hours) spent in the review meeting
- Average preparation rate - the average rate at which each reviewer prepared, e.g., pages per hour, which indicates how quickly the material was reviewed.
- Meeting rate, e.g., pages per hour
- (Major) defect detection rate – the ratio of the number of defects found per the size of the work product, e.g., defects per page or defects per 1000 SLOC (KSLOC)

After a sufficient amount of peer review data has been collected by similar projects and data integrity has been established, average rates can be established for the 3 preparation, meeting, and defect detection rates (for each type of peer review for each type of work product). This is usually done by the organization's measurement guru or analyst. From these averages high and low detection rate thresholds can be established to trigger further analysis of the data and possible action. An advanced method is to apply statistical process control (SPC) [25] and calculate the normal range of performance for each of these rates.

2. Process Based Defect Taxonomy

Process based taxonomy assumes that defects are recorded when they are found throughout the software process lifecycle, including their classification according to the defect taxonomy. Recording defects and classifying them can help understand the process with respect to all those activities that produce defects in particular, they help in identifying process weaknesses (high-defect steps). In the other view, after product testing a list of defects is produced that pinpoints the defects in product and their roots causes. This view is not talking about the process as a root to defect and usually brings up suggestions for rework approach and believes that the kinds of defects may point out the best approach for doing rework (e.g. direct repair, review, redesign, retest, etc.). Defect characteristics of artifacts may help with risk assessment for process decisions such as task priorities, go/no-go decisions, reimplementation decisions etc.

Despite high amount of effort into studying root cause analysis or qualitative analysis and bringing up "Why" a specific defect is "produced" and how to prevent it in future products, in most of the studies the importance of process has been ignored. if a taxonomy comes up with processes as center of gravity, and point out the probability of preventing defects based on process developments, that not only will help the future projects – as the current process based taxonomies do – but also serves to make a better final product for an ongoing project.

2.1. Defect Driven Process Taxonomy

One of the biggest contributors to software defect is "human" factor that is usually underestimated by others. It seems like that they focused on the results of human actions rather than "nature" of human behaviors that will lead to those results.

A study is needed with focus on defects based on the process in which they are found. These defects are selected based on their potential impact on the final product.

This study should focus on the following characteristics of the process:

- When process starts and when ends
- What are the major and minor goals of the process
- How human behaviors can affect the process and in which level
- Is it possible to break down the process to sub-processes

- Define the defects that can be produced by specific process
- Define the severity of those defects and their affect in project success
- Use mathematical methods for modeling the process

Then a process improvement method can be elicited from this study. The mathematical methods will then be deployed to help in quantifying the processes in order to make them more manageable and understandable.

2.2. Framework

Generally to the authors believe the process based taxonomy framework should focus on the following:

- Process Attributes;
  - Taxonomy is defined by attributes. Each attribute has a set of values. The values represent defect characteristics that must be registered at the process studying procedure.
  - These attributes should be the ones which can help analyze the process in future.

- Process Structure;
  - Structure defines the relationship between processes and the way they interact with each other.
  - One of the most used structures can be orthogonal relationship that has been vastly used by IBM.

- Process Properties;
  - Unique properties of each process that can be helpful for identifying that process as a "type" and be used in future cleaning procedures to make it faster and more effective should be defined for each process.

- Effectiveness rate;
  - A touchable parameter to evaluate each cleaned process is needed to be in place that will be used in order to compare the results.

Finally a modeling method can combine the entire information gathered through this procedure to introduce software defect taxonomy. This taxonomy could prove to be much more cost effective than those already exist.

Conclusion

Much of the published work in the defect Taxonomy modeling area is focused on debugging rather than defect prevention and mostly they emphasis the final product or testing results. This review paper shows that many past studies have suffered from including the process as a standalone artifact. The issues and problems surrounding the models illustrate how difficult defect prediction is and how easy it is introduce modeling errors. Specifically, we found out that those existing models using size and complexity metrics alone are incapable of predicting defects accurately. Furthermore, these models do not describe how defect introduction and detection variables affect defect quantity and quality. The existing software defect taxonomies often do not focus fully on the process based approach. In most cases process and product are studied in parallel and significant amount of time is spent to identify and debug the defects or prevent them from occurring again. To the author's believe, considering a process based taxonomy which is carefully created and implemented benefits from strength of already developed taxonomies and avoid their weaknesses.

References:

[1] Douglas A. McIntyre, 17 Feb 2010 *"24/7 Wall Street"*, http://247wallst.com/

[2] Wikipedia "List of Software Bugs", http://en.wikipedia.org/wiki/List_of_software_bugs

[3] Gleick James, A bug and a Crash, 1996 *"Sometimes a bug is more than a nuisance"*. http://www.around.com/ariane.html

[4] Softwareengineeringrefrence.com, Software Failure list, 2010 http://www.sereferences.com/software-failure-list.php

[5] National Institute of standards and technology NIST, WED, APR 28, 2010 14:59 http://www.itbusinessedge.com

[6] Byline Bill Hoffman, CTO, Kit ware, 2009 "Software defects, one developer at a time"

[7] Lim Joanne October 1998, "An Engineering Disaster: Therac-25" PP 1-2

[8] Boehm Barry, victor R. Basili , Jan 2001 "Software defect reduction top 10 List" http://portal.acm.org/citation.cfm?id=621640

[9] Norman E. Fenton, 1999, "A Critique of Software Defect Prediction Models"

[10] Freie Universitat Berlin Researchhome website https://www.inf.fu-berlin.de/w/SE/DefectTaxonomy

[11] Chillarege Ram, 1992, "Orthogonal Defect classification a concept for in-Process Measurement" http://www.chillarege.com/articles/odc-concept

[12] Joe Schofield from Sandia National Laboratories, 2005 , "Beyond Defect Removal: Latent Defect Estimation with Capture Recapture Method (CRM)", PP 1-2

[13] IBM. Refer to IBM Center of software Engineering at IBM Research web site ODC. http://www.research.ibm.com/softeng/ODC/ODC.HTM

[14] Raymond Madachy and Barry Boehm from university of southern California, 2001, "Bayesian Analysis of Software Costs and Quality Models"

[15] http://csse.usc.edu/tools/odc_coqualmo.php

[16] Madachy R., Boehm B., Richardson J., Feather M., Menzies T." Value-Based Design of Software V&V Processes for NASA Flight Projects, In: AIAA Space 2007 Conference, (2007)", http://csse.usc.edu/csse/TECHRPTS/2008/usc-csse-2008-830/usc-csse-2008-830.pdf

[17] Pavankumar Pothuraju's weblog, 23 Aug 2007, "The library of software Testing" http://geekswithblogs.net/ppothuraju/Default.aspx

[18] Shaik Mohammad Rafi ,Dr. K. Nageswara Rao and Shaheda Akhtar, 2002, " Software Reliability Growth Model with Logistic-Exponential Test-Effort Function and Analysis of Software Release Policy". http://www.enggjournals.com/ijcse/doc/IJCSE10-02-02-50.pdf

[19] Chillarege Ram , 1993, "Process Measurement, Analysis and Control" http://www.chillarege.com/articles/odc-process-control

[20] Karl E. Wiegers, Peer Reviews in Software, Addison-Wesley, 2002, "Using Peer Review Data to Manage Software Defects" http://www.compaid.com/caiinternet/ezine/lett-defects.pdf

[21] Beth Chrissis Mary , et al., 2007 "CMMI ® for Development, Version 1.2, Addison Wesle".

[22] Ronald A. Radice, "High Quality Low Cost Software Inspections, Paradoxicon Publishing", 2002

[23]H. Lett Steven, 2002, "Using Peer Review Data to Manage Software Defects" http://www.compaid.com/caiinternet/ezine/lett-defects.pdf

[24] H. Lett Steven 2001, "General Management of Software Defects" PP 4-5

[25] *Addison Wesley*, 1999, " Measuring the Software Process" PP 3

[26] Hower Rick, 2007, "software QA and testing frequently asked questions" http://www.softwareqatest.com/qatfaq1.html

[27] Raymond Madachy, Barry Boehm, USC-CSSE-2008-817, "ODC COQUALMO - A Software Defect Introduction and Removal Model using Orthogonal Defect Classification" ,

Enhanced Economic Modeling for Software Quality Failure Analysis

Dinesh Kumar Saini[1] and Moinuddin Ahmad[2]
1Fauclty of Computing and Information Technology, Sohar University, Sultanate of Oman
2 Faculty of Business, Sohar University, Sultanate of Oman
dinesh@soharuni.edu.om, mahmad@soharuni.edu.om

Abstract— **In this paper an attempt has been made to study the defects and failures in the software quality and cost related to software development. Managing quality is a costly affair and it is a major concern in the software development environment. Quality of software is a very important factor for selling software. Better quality requires better testing. This testing requires manpower and money resources. So performing defect analysis, applying defect detection technique and removing faults for higher gains in reliability with reasonable cost is the ultimate requirement in the software development environment.**

I. Introduction

A defect arises in software due to problems in the development process. By continuous reviews defect can be prevented in software development process [1]. If defects are present in software they can be identified through testing. After identification of defect rework is required for its removal. Rework requires resources that are manpower and money. If defects are not removed before release of software then it will lead to the failure of software. The costs for removal of defects are very small compared to compensating failure of software.

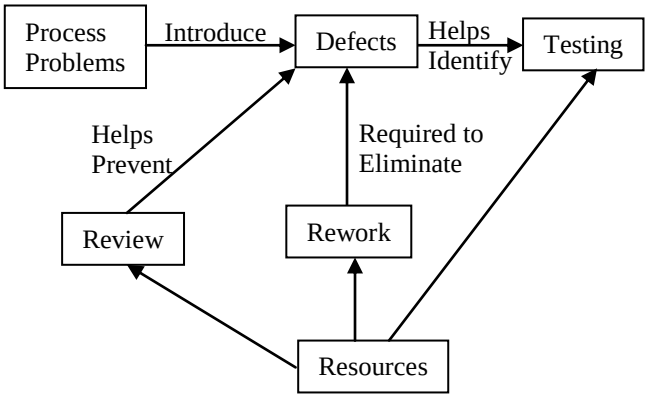


Fig 1: Concept Map for Defect Analysis

So defects should be removed before release of software. But more testing causes more use of resources so we also need a optimum testing period for which software has to be tested.

Dinesh Kumar Saini is Associate Professor at Faculty of Computing and Information Technology, Sohar University, Sohar, Sultanate of Oman (dinesh@soharuni.edu.om)
Moinuddin Ahmad is Professor at the Faculty of Business, Sohar University, Sultanate of Oman (mahmad@soahruni.edu.om), Sultanate of Oman.

But by increasing the testing period, consumption of resources also increases, which results in cost escalation, hence a decision has to be taken for the optimum testing period for which the software has to be tested.

II. Software Quality Perspective

Quality is highly needed in software development environment. Following are the software quality consideration known as Five-Star Quality perspective which is must in the software development [21]:

1. Customer Satisfaction:
   - Conformance to requirements/needs of the customer
   - Satisfaction Levels of the customers
   - When & where to use these techniques
2. Value to stack-holders and business owners
   - Profitability
   - Competitive advantage over each other
   - Personnel satisfaction and acceptance
3. Properties ("utilities") which are must
   - Maintainability
   - Complexity
   - Usability
   - Reliability(MTBF- Mean time between failures)
   - Portability
4. Defectiveness in the product and process
   - Defect levels
   - Defect severity
   - Defect removal
5. Development Effectiveness for product and process
   - Schedule & budget performance
   - Features & functions delivered
   - Process quality

III. Why Defect Removal before Software Release

The cost for defect removal is very less compared with the external failure of software. This can be seen from the following example. CoSQ (Cost of Software Quality) analysis of a recent Enterprise Integration (EI) Debacle [21, 22].So from the table 1 we can conclude that the fault removal before software release is cost effective for any software.

IV. Types of Cost

Cost of quality for any software to make it more reliable is an area under research. This cost is associated with preventing, finding and correcting defective work. The basic PAF (Prevention Appraisal Failure) model for cost of

quality derived from the manufacturing industry that has been repeatedly used in software quality is given here[6]. Following are the classification for costs of quality:

A. Conformance (control) cost

The conformance costs comprise all costs that need to be spent to build the software in a way that it confirms to its quality requirements. These costs are divided into internal failure cost and external failure costs.

Table 1

Major Category	Subcategory	Cost summary	Comments and explanation
Non-conformances	External	$ 700 M+	Loss and damage related to company
	Internal	~ $ 50M	Rework cost and cancellation cost
Level Appraisal	Discovering the condition	~ $ 5 M	After the fact testing, no real Software Quality Assurance
	Assuring the achievement	< 1 M	Several aborted release attempts based on hits and trials
Prevention and under quality	Project/process interventions	~0	Poor process and collects only a few metrics
	Quality basis management	~0	Poorly release criteria for acceptance testing and , no quality standards, fuzzy quality goal definition not properly designed

B. Prevention cost

Costs of means to prevent the injection of faults, which incurs cost on following process:

- Requirements
  - Marketing Research
  - Customer/User survey
  - Risk Analysis
  - Prototyping
  - Requirements/Specification Review
  - Scheduling
- Project
  - Quality Planning
  - Process Validation and Reassessment
  - Platform and Tools Quality Assessment
  - Developer Quality Training
  - Quality Metrics Data Collection
  - Design For Quality
  - Inspections/Reviews
  - Configuration Management
  - Change Management
  - Supplier Assessment
- Conformance to Design Specifications
- Configuration management
  - Platform and Tools
  - Training
- SQA (**Scottish** Qualifications Authority)administration
  - Process Standards, Procedures, and
  - Definitions
  - Data Maintenance, Analysis and
  - Reporting
  - Planning
  - Documentation of Plan Deviations
  - Technical Reviews
  - Reporting
  - Auditing
  - Training
  - Process Improvement

C. Appraisal cost:

The appraisal costs are caused by performance of various types of test and reviews [18]. That is as follows [7]:

- Verification and validation
- Test Planning
- Test Data Generation
- Test Execution
- Test Result Logging
- Code Inspections
- Statistical Analysis
- Acceptance Testing
- Test Traceability to Requirements
- Reporting

The appraisal cost can be further classified into two parts:
Setup cost: Cots for purchasing testing tools, configuring the test environment and so on.
Execution cost: The cost associated with the actual test execution or review meetings and mainly personnel costs.

D. Non-conformance cost

The nonconformance costs occur when the software does not confirm to the quality requirements of the software. These costs are divided into two categories, internal failure cost and external failure costs.
Internal failure costs: It is caused by failures that occurred during development of project. It includes:

- Product Design Defect Costs
  - Causal Analysis and Reporting
  - Pervasive Design Modification
  - Rework and Retest
  - Work Products Wasted or Redesigned
- Implementation Defect costs
  - Defect Measurement and Reporting
  - Defect Isolation and Correction
  - Causal Analysis and Reporting
  - Process Correction
  - Repair Inspection
  - Retest and Integration
  - Reduced Productivity

External failure cost: the costs that result from failures at the client site or at the customer site which are described as follows

- Lost Sales
- Lost Market Share
- Litigation and Liability Costs
- Loss of Customer Goodwill
- Catastrophic Loss

- Technical support
  - Maintenance Release Due to Defects
  - Defect Notification
  - Warranty Work
  - Increased Change Requests

Fault removal cost is attributed with external failure cost and internal failure costs which is the non conformance side. All the failures results into cost which is required to fight with the failure and the failure removal cost is independent of the faults whether the fault occurred internally or externally. Attempts are always made in the software development environment to avoid failures.
Code inspections are carried out in the software development environment so that faults can be removed which has not ended with failures. Removal cost is different irrespective of the technique used. When a failure is detected by a test identifier considerable efforts are required for handling that fault. Inspections also help to find the fault directly. Fault removal efforts are costly and it also contains cost for re-testing and re-inspections. External failures also lead to support cost.

These are all costs connected to customer are, especially the effort from service workers identifying the problem. Finally, compensation costs could be part of the external failure costs, if the failure caused some kind of damage at the customer site. We might also include loss of sales because of bad reputation in the external failure costs [18].

V. Failure Analysis

The money and cost related to quality is time dependent and time is very important dimension. The time of arrival or detection of fault and estimation of the removal cost is dependent on time. The removal cost also used for the basis for the calculation of the present net values [2, 3]. The first dimension is important because the additional work increase for the removal, the later the fault is removed which also increase the cost.

For example if a design fault is detected early only the design document need to be changed, price for removal is also less and later also the code and test cases have to be adapted according to the changed design document. This also shown, for example, [21, 22, and 23] the later is important because to be able to analyze in dependence of the point in time when they occur. We depict a typical change in non-conformance costs during time in fig 2 [12, 13, 14, and 15]. The failure rate is defined as the total number of failures within an item population, divided by the total time expended by that population, during a particular measurement interval under stated conditions.

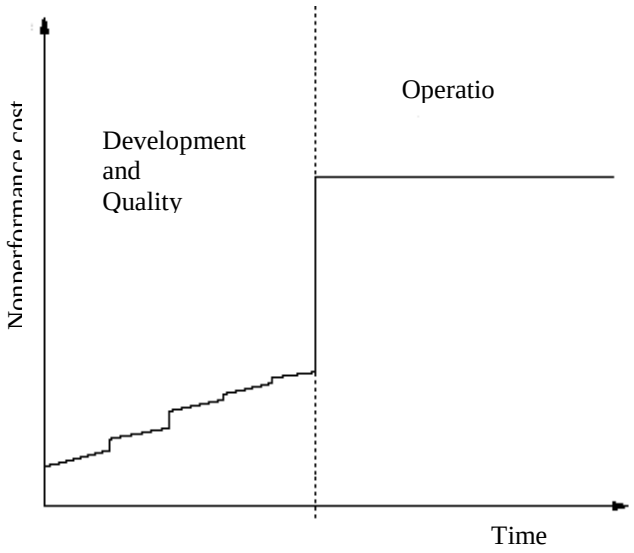


Fig. 2 Dynamics of nonconformance costs/ and failure over time

VI. Cost Models

For calculating ,the quality cost, the net present value of the cash flows (NPVCF), is being considered i.e., the revenues and costs during the life cycle of the software that are related to quality, in this paper. This metric is adequate to judge the earning power and also to evaluate different defect-detection techniques. For further analyses other metrics, such as Return on Investment (ROI) or Software Quality Program Instruction (SQPI) can be used [9, 25].

A. Direct Measurable Cost

Those costs that are directly measured or at least can be estimated accurately during the usage of the defect-detection techniques are termed as direct measurable costs. For a specific defect-detection technique we have some fixed initial investments which are known as setup costs (C_{Setup}) that contain for example the costs of tools or workstation. We assume them to appear at the start of software development and therefore they already have the net present value. Furthermore, we have the dynamic part of the appraisal costs, e.g. personnel for tests and inspections, that we call execution costs (C_{exec}). The fault removal costs for faults that were found during the defect-detection technique under consideration are directly measurable. It is called (C_{remv}), pr which denotes the fault removal costs at the period pr [5, 6].

Having this information we can determine the direct costs (C_{direct}) of the defect-detection technique by adding up the execution and fault removal costs for all periods where we have direct measurements and the setup costs. The periods that we look at here are typically the periods in which the defect-detection technique was used. The costs now need to be discounted to the net present value using the discount factor D that represents the average cost of capital [6].

Calculation for direct cost (C_{direct}):

$$c_{direct} = c_{setup} + \sum_{pe=0}^{ne} \frac{c_{exec(pe)}}{(1+D)^{pe}} + \sum_{pr=0}^{nr} \frac{c_{remv(pr)}}{(1+D)^{pr}} \quad \ldots 1$$

In equation (1) calculation for execution cost(cexec (pe))has done on regular period basis.
There are three different cases on the basis of payment made for the fault removal.

Case (1): when the payment made for the fault removal is immediate after each fault removal at time (pr) and independent from the time (pe). Then equation (1) is used.

Case (2): when payment made for the fault removed is in regular period basis along with execution cast then

$pe = pr = p$ and $ne = nr = n$ and then equation (1) becomes

$$c_{direct} = c_{setup} + \sum_{pe=0}^{n} \frac{c_{exec}(p) + c_{remv}(p)}{(1+D)^{p}} \quad ..2$$

Case (3): with the consideration that the fault removal takes considerable time and the cost of removing an error is increasing as the number of errors removed is increasing. Then expected total cost for the fault removed during the period [0, pr) is given as follows

$$E2(pr) = m(pr)[c + \left(\frac{c_w}{2}\right) * m(pr) \quad ....3$$

where,
c_w = expected value of variable w
w = random variable of cost to remove errors during testing
m(pr) = expected no. of software fault by time pr.
c = deterministic cost to remove each error unit time during testing phase.
With this consideration c_{direct} is given as follow:

$$c_{direct} = c_{setup} + \sum_{pe=0}^{ne} \frac{c_{exec}(pe)}{(1+D)^{pe}} + \frac{E_2(pr)_{remv}}{1+D)^{pr}} \quad ....4$$

B. Prediction Using Reliability Models

Defects are distributed in software randomly. By using any good reliability model we can predict the defects in software for particular time interval. Let we are using Poisson's non-homogeneous process for predicting number of defects in software in period p that is proposed by others like Goel and Okumoto model[11].
Suppose that a piece of software under testing is subjected to failure at random times because of bugs in the code. Let {N(T), T>=0} be a counting process that represents the cumulative no of failures by time T, where T is the running time of the program. The time-dependent error detection rate model makes the following assumption about N(T). [7,8].
(a) there are an expected number of bugs a that will be discovered over the life of the software.
(b) the number of bugs that are discovered during distinct testing periods are independent
(c) the expected number of bugs discovered in a small interval (T, T + p) is approximately proportional to the expected number of errors remaining at time T and the length of interval p; thus

$$f(p) = E\{N(T+p)\} - E\{(N(T)\} = b[a - E\{N(T)]p + o(p)$$

under this assumption, N(T) is a non-homogeneous Poisson process with mean value function $\Lambda(T)$,

$$P\{N(T) = n \mid a, b\} = \frac{\Lambda(T)^N}{n!} e^{-\Lambda t} \quad ....6$$

where,

$$\Lambda(T) = E\{N(T) \mid a, b\} = a\{1 - e^{-\Lambda t}\}$$

a= expected number of bugs,

b=the rate of discovery of bugs

Gaffney (1984), had suggested several empirical formulae for the number of bugs per line of code, for different languages, for code containing S lines, a=0.021 S and S=4.2+0.0015 S4/3 are the two proposed [11].
Note that many models use execution time whereas the periods are measured in calendar time. Hence, a conversion might be necessary.

After knowing the number of failures the fractions of severity and estimates of costs per severity class are needed. They are used to determine the number of the failures belonging to different severity class. Then the number can be multiplied with the estimated costs per severity class.

This gives as the following equation for the future costs[6].

$$c_{fut} = \sum_{i=n}^{u} \frac{\sum_{s=1}^{S} f(i)p(s)c_{ext(s)}}{(1+D)^{i}} \quad ....7$$

where n is the period in which we start the prediction, u is the upper limit of the prediction periods, S is the highest severity class, P(s) is the fraction or probability of severity class s, and c_{exec}t(s) are the estimated external costs for a failure of severity class s.

C. Estimation

Finally, we would like to estimate the revenues (*profits)* that we will have after using the defect-detection technique. These are the external failure costs that we saved by finding and removing faults before releasing the software in the market [16]. Therefore we can use the same cost categories as above, i.e. fault removal, support, and compensation costs. The main difference is that we do not have an empirical basis to estimate the occurrence of the faults. Therefore, we use expert opinion to estimate the mean time to failure (MTTF) of each fault and the severity of the corresponding failure. All the revenues *(profits)* have to be discounted to the present value. For this we need the discount factor D again that represents an average cost of

capital. It is difficult to estimate the time until an external failure will occur and will result in further costs. This can be estimated based on the MTTF estimated earlier. All this assumes that we have a fixed granularity for the periods that is used also for the estimate of time periods for external failures. The severity is finally used again to estimate compensation costs. Based on this the following equation can be used to calculate the net present value of the revenues(*profits)*. [18, 19, 20]

$$r = \sum_{i=n}^{u} \frac{c_{remv}(i) + c_{sup}(i)\, c_{comp}(i)}{(1+D)^i} \quad \ldots..8$$

where n is the period after the usage of the defect-detection technique, u is the upper limit if the estimation periods, c_{remv} are again the fault removal costs, c_{sup} the support costs, and c_{comp} possible compensation costs.[18,19,20]

There are two criteria of making decision on when to stop testing (deciding optimum value of n) [10]:

1. when the reliability has reached a given threshold,
2. when the gain in reliability cannot justify the testing cost.

Following are the effecting factors to the value of n. [9,10] to make r high.

1. If people put more weight on the testing cost and believe that the testing cost is the main concern, then they would like to end up do testing earlier.
2. increasing the error removal cost coefficients will result in a shorter testing time.
3. if reliability is a critical concern, people need to spend more time doing testing.
4. if the risk cost is the main concern, people spend a longer amount of time doing testing.
5. increasing the mission time x will result in a longer testing time, since in order to have the software survive for a longer mission time without failure, one has to take a longer amount of time doing testing.

By considering above reasons we have to try for optimal value of n that will lead to more benefit.

D. Quality Economics

With collected and established metrics the equation is formulated for computing quality cost and quality economics using defect detection method [27, 28]. Direct measurable costs are used for computing the defect-detection technique usage, and it also used for the predicting future costs from the failure behavior of the software. Revenue is estimated based on Mean Time to Failure (MMTF) and severity estimates for the found faults. The given equation is used for the net present value of the cash flows [2, 4].

$$NPVCF = r - c_{direct} - c_{fut} \quad \ldots.9$$

Net present value of the defect detection is computed by subtracting the direct measurable costs and the future costs from the revenues to get the net present value of the defect-detection technique which represents the benefit from the usage of the technique.

V. Conclusion

In this paper initially we introduce the concept of quality and defect detection techniques. Software quality perspectives are studied in detail that leads to cost of development in the software development environment. We are suggesting that defect removal before release of software always reduce the cost of software development. All types of costs related to software development process are analysed related to defect removal. Formulations for software quality economics were carried out and analysed for the failure analysis. It is found that defects leads to cost increase and detection of defects in early life cycle before release reduce cost to some extent. Managing quality is costly affair and it is major concern in the software development environment. We are suggesting that quality of software is very important factor for selling software in the competitive market. . Better quality requires better testing. This testing requires manpower and money resources. So we found out that performing defect analysis, applying defect detection technique and removing faults for higher gain in reliability with reasonable cost is the ultimate requirement in the software development environment.

REFERENCES

[1] C.Jones. Applied Software Measurement: Assuring Productivity and Quality. McGraw-Hill, 1991.

[2] Champ, C. M., and S. -P. Chou (2003). "Comparison of Standard and Individual Limits Phase I Shewhart, *X, R,* and *S* Charts," *Quality and Reliability Engineering International*, Vol. 19(1), pp.161-170.

[3] Chan, L. K., S.W. Cheng, and F.A. Spiring (1988). "A New Measure of Process Capability: C_{pm}," *Journal of Quality Technology*, Vol. 20(3), pp. 160-175.

[4] Cruthis, E. N., and S. E. Rigdon (1992-1993). "Comparing Two Estimates of the Variance to determine the Stability of a Process," Quality Engineering, Vol.5(1), pp. 67-74.

[5] Grigg, O., and D. Spiegelhalter (2007). "A Simple risk-Adjusted Charts," *Journal of American Statistical Association*, Vol. 102, pp. 140-152.

[6] Dinesh Kumar Saini, Nirmal Gupta "Class Level Test Case Generation in Object Oriented Software Testing, International Journal of Information Technology and Web Engineering, (IJITWE) Vol. 3, Issue 2, pp. 19-26 pages, march 2008.

[7] Kang, L., and S.L. Albin (2000). "On-Line Monitoring When the Process Yields a Linear Profile," *Journal of Quality Technology*, Vol. 32(4), pp.418-426. Available at www.asq.org/pub/jqt.

[8] Kevin McDaid, "Deciding how long to test software", The Statistician (2001), 50, Part2, pp. 177-134.

[9] Kim, K., M. A. Mahmoud, and W. H. Woodall (2003). "On the monitoring of Linear Profiles," *Journal of Quality Technology*, Vol. 35(3), pp.317-328.

[10] Krasner, H., "Using the Cost of Quality Approach for Software Development", Cost of SW Quality 22(22).

[11] Krasner, H., "Using the Cost of Quality Approach for Software", CROSSTALK The Journal of Defense Software Engineering, 11(11) November 1998.

[12] M. Li, C. Smidts and R. W. Brill, "Ranking Software Engineering Measures Related to Reliability Using Expert Opinion" IEEE 2000.

[13] Montgomery, D.C. (1999). "Experimental Design for Product and Process Design and Development" (with commentary), *Journal of the Royal Statistical Society Series D (The Statistician)*, Vol.48 (2), pp.159-177.

[14] Montgomery, D.C. (2009). *Design and Analysis of Experiments*, 7th ed., Wiley, New York.

[15] Montgomery, D.C. (2009). *Introduction to Statistical Quality Control*, 6th ed., Wiley, New York.

[16] Montgomery, D.C., C. L. Jennings, and M. Kulahci (2008). *Introduction to Time Series Analysis and Forecasting*, Wiley, New York.

[17] Montgomery, D.C., and G. C. Runger (2007). Applied Statistics and Probability for Engineers, 4th ed., Wiley, New York.

[18] Pignatiello, J. J. ,Jr., and T. R. Samuel (2001). "Estimation of the Change Point of a Normal Process Mean in SPC Applications," Journal of Quality Technology, Vol.33 (1), pp. 82-95.

[19] R.L. Upchurch and Ann E. Kelley Sobel, "STRUCTURAL ASSESSMENT OF COST OF QUALITY", 32nd ASEE/IEEE Frontiers in Education Conference, November 6-9, 2002 Bostan, MA.

[20] Ross, S. M. (1971). "Quality control Under Markovian Deterioration," *Management Science*, Vol.17 (9), pp.587-596.
[21] S. Slaughter, D. Harter, and M. Krishnan. Evaluating the Cost of Software Quality. Communications of the ACM, 41(8):67-73, 1998.
[22] S. Wagner and T. Seifert "Software Quality Economics for Defect-Detection Techniques Using Failure Prediction", 3-WoSQ '05, May 17, 2005, St Louis, Missouri, USA.
[23] S. Wagner "Towards Software Quality Economics for Defect-Detection Techniques" In Proc. 29th Annual IEEE/NASA Software Engineering Workshop, 2005.
[24] Testik, M. C., C. M. Borror (2004). "Design Strategies for the Multivariate EWMA Control Chart," *Quality and Reliability Engineering International*, Vol. 20, pp. 571-577.
[25] Woodall, W. H. (2007). " Profile Monitoring," entry in *Encyclopedia of Statistics in Quality and Reliability*, edited by Fabrizio Ruggeri, Frederick Faltin, and Ron Kenett, John Wiley & Sons, New York.
[26] Zhang. X. and H. Pham, "Software release policies with gain in reliability justifying the costs", Annals of Software Engineering 8 (1999) 147-166.
[27] Dinesh Kumar Saini "Testing Polymorphism in Object Oriented Systems for improving software Quality" ACM SIGSOFT Volume 34 Number 2 March 2009.
[28] Dinesh Kumar Saini, Wail M. Omar "Software Testing For Semantic Service Oriented Architecture for E-Health Software Services" SERP'10 - 9th international Conference on Software Engineering Research and Practice (USA) http://www.world-academy-of-science.org/, P.No. 240-246.

New Trials on Test Data Generation: Analysis of Test Data Space and Design of Improved Algorithm

So-Yeong Jeon[1] **and Yong-Hyuk Kim**[2,*]
[1]Department of Computer Science, Korea Advanced Institute of Science and Technology
Daejeon, Korea, E-mail: presentover@gmail.com
[2]Department of Computer Science and Engineering, Kwangwoon University
Seoul, Korea, E-mail: yhdfly@kw.ac.kr
* Corresponding author

Abstract – *Test (input) data generation is important for the algorithms/software/hardware testing. The previous researches on test data generation motivate us to find some meaningful information of generated test data. In this paper, we differentiate the test (input) data space (i.e., problem instance space) from the output data space (i.e., solution space), by examining the test data generated in terms of optimality (one of the performance measures). We investigate the problem instance space of the 0/1 knapsack problem, by calculating some kind of cost-distance correlation; the correlation was quite different from those in well-known combinatorial optimization problems. Also, we improved a greedy algorithm of the 0/1 knapsack problem by generated test data. The improved algorithm showed superiority to the original one under 10,000 random instances. This paper presents some promising values of the researches on the test data space and the test data generation for improving the tested module.*

Keywords: Problem instance space, 0/1 knapsack problem, greedy algorithm, algorithm performance analysis, cost-distance correlation.

1 Introduction

Test data generation or finding specific test data has been interest of many researchers in various fields. For the hardware testing, Saab *et al.* [1], Srinvas *et al.* [2], and Rudnick *et al.* [3] tried to generate test data finding some faults of sequential or combinatorial circuits. Corno *et al.* [4] tried to generate a test bench for a microprocessor design validation in terms of statement coverage. About software test data generation, McMinn surveyed those trials [5] for structural testing, non-functional testing and grey box testing. In fact, the test data generation is an undecidable problem in general. In the test data generation fore-mentioned, in fact, researchers use some ideas based on the meta-heuristic search algorithms. Then one of the important things is what objective function is used for the test data. But the objective value calculation in some case depends on the hardware environment. Execution time is the one of the examples (refer to [5] for the worst case execution time testing).

Therefore it would be general approach to define objective function in more abstract manner so that the calculation of objective value is independent from the hardware environment. *We regard a module of the software/hardware being tested as an algorithm.* The meaning of test data becomes the best/worst case defined for given testing objective function (e.g., the number of covered statements, the number of basic operations, or the degree to which the output is close to optimal one). The worst/best case is just some test data maximizing/minimizing the objective value. In the field of algorithm studies, we can see those researches. Johnson and Kosoresow [6] tried to find the worst case, in terms of optimality, of algorithms for online Taxicab problem. Cotta and Moscato [7] tried to find the worst case of the shell sort to approximate the worst-case time complexity. Hemert [8] tried to find the worst case of some algorithms for binary constraint satisfaction, Boolean satisfiability, and the travelling salesperson problem, in terms of time complexity. Jeon and Kim [9] tried to find the worst cases of well-known internal sorting algorithms as well as shell sort. They also tried to find the worst case, in terms of optimality, of algorithms for the 0/1 knapsack problem and the travelling salesperson problem.

Above researches give us some questions; this paper deals with two of those questions considering the 0/1 knapsack problem and the greedy algorithm as tested algorithm, as an extension of [9]. (From now on, the problem instances have the same meaning as the test data and we call problem instances just instances) The first question is, how different is generating test data of algorithms from finding solutions of problems? As the previous researches have focused on the cost-distance correlation (CDC) of solution space, we will calculate cost-distance correlation to investigate problem instance space, which is the space of the test data (Section 2). This will show how the test data space can be different. Note that, to the best of our knowledge, we are the first researchers using CDC for the investigation of *problem instance* space. The other question is, can we use the performance test (input) data for improving the tested algorithm? Previous researches did not focus on this question. Someone may underestimate the importance of the test data and want to see just the per-

formance score. We will show the input data itself can be used for finding defects of the tested algorithm (Section 3). We observe the test data for which the tested greedy algorithm gives a far-from-optimal solution. Based on the observations, we devise an improved algorithm in terms of optimality. We make conclusions in Section 4.

2 Analysis of Test Data Space

```
U = list of all the given items;
W = given capacity of knapsack;
K = Ø; // items to be in the knapsack
Sort U as an array by profit density in non-increasing order†;
for each item_i ∈ U
{
  if ( weight sum of K + weight of item_i ≤ W )
    Add item_i to K;
}
return K;
```

Figure 1. Pseudo-code for tested greedy algorithm

†Put the lighter item at lower index when the same profit density occurs.

```
// (problem) instance is given as an array of items
for each i-th item
{
  // Let the i-th item be '(v,w)'; v is its value and w is its weight
  Consider its eight neighbors†: (v+1,w), (v−1,w), (v,w+1),
  (v,w−1),(v+1,w+1), (v+1,w−1), (v−1,w+1), and (v−1,w−1) ;
  Take the one improving the quality of the instance best;
  Update (v,w) to the new one;
}
```

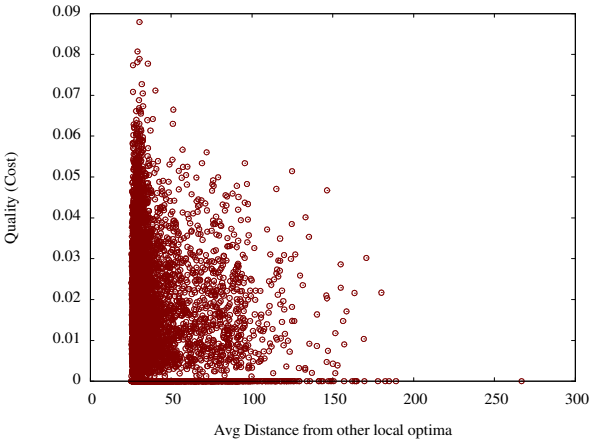
Figure 2. Local optimizer for 0/1KP instances

†We exclude any item of which the value or the weight is not in [1,100].

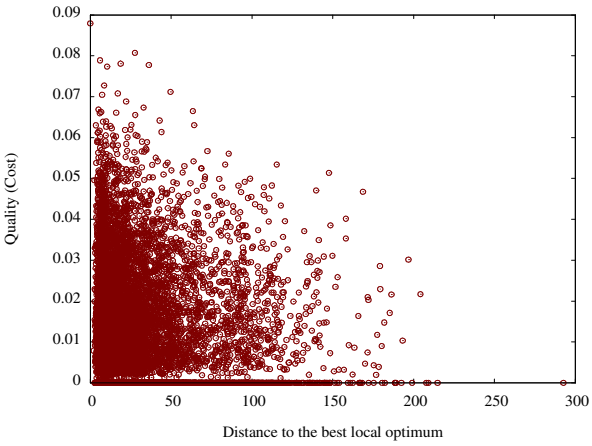
This section deals with the problem instance space of the 0/1 knapsack problem (0/1KP). In 0/1KP, we are given a knapsack and items. The knapsack has the capacity and each item has its own value and weight. The objective of 0/1KP is to choose items making their value sum be the highest, at the same time the weight sum not exceed the capacity of the knapsack.

We take as the tested algorithm the greedy algorithm which was dealt with in [9] (see Figure 1). The algorithm first sorts the given items by their profit density (value per unit weight) in non-increasing order. The lighter item is put first when the same profit density occurs. The algorithm tries to choose items in the sorted order. If the chosen item does not exceed the capacity of the knapsack, it is put into the knapsack. Otherwise, the algorithm tries to choose the next item according to the sorted order. The algorithm exits after considering the last item in the order.

This algorithm does not give an optimal solution in every case. As in [9], we define the quality of a problem instance as $1-P/O$, where P is the objective value of the solution obtained by the tested algorithm (the greedy algorithm fore-mentioned in this paper), O is the objective value obtained by the optimal algorithm (e.g., based on dynamic programming). In terms of optimality, an instance is the worst case if the quality is the highest. (This means the tested algorithm gives far-from-optimal solution for the instance.) The best case has the lowest quality.



(a) *dOthers* vs. quality



(b) *dBest* vs. quality

Figure 3. Cost-distance correlation (20 items, weight coefficient 0.5)

The method of getting cost-distance correlation is similar to one for the graph bi-partitioning problem given in [10]. Fore-mentioned 'quality' plays the role 'cost' plays in cost- distance correlation. In the search space, all the problem instances have the same number of items and the same weight coefficient. Here the weight coefficient is a real number in (0, 1) and determines the capacity of the knapsack by $\lfloor$(*weight coefficient*)×(*weight sum of all the given items*)$\rfloor$. ($\lfloor . \rfloor$ means "the floor of".) Weights and values of items are limited to be integers in [1,100]. Observing the above constraints, we generate 10,000 instances randomly and take the local optimum of each by using the algorithm in Figure 2. Then we remove any redundant local optimum. For each local optimum, we calculate two kinds of numerical values. One is the average distance from all other local optima (we call it '*dOthers*'). The other one is the distance to the best local optimum (we call it

Table 1. Some statistics obtained from the method of [10] (# of local optima are 10,000)

#items	Weight coefficient	best Q	avg Q	std Q	max D	min D	avg D	std D	Corr(*dOthers*,Q)
10	0.25	0.389	0.035	0.054	271.708	0.346	23.185	24.241	–0.052
	0.5	0.207	0.019	0.031	274.740	0.247	23.160	24.202	–0.037
	0.75	0.156	0.010	0.019	271.610	0.234	23.169	24.216	0.010
20	0.25	0.176	0.021	0.024	301.207	0.858	37.670	30.634	–0.045
	0.5	0.088	0.012	0.014	304.889	0.924	37.591	30.587	–0.036
	0.75	0.058	0.006	0.008	301.276	0.884	37.677	30.631	–0.019
30	0.25	0.081	0.016	0.014	332.748	1.447	49.532	34.798	–0.035
	0.5	0.050	0.009	0.008	332.547	1.570	49.542	34.845	–0.019
	0.75	0.033	0.004	0.005	326.248	1.531	49.628	34.873	–0.015

'Q' means the quality, 'D' means distance. All statistics are from local optima. Local optimizer operates toward increasing the quality of the given instance. Corr(*dOthers*, Q) is the correlation coefficients between *dOthers* and quality.

'*dBest*'). For getting the distance, we regard a problem instance as an array of items, thus each item can be indexed. The definition of the distance is based on the Manhattan distance. We calculate the distance between *A* and *B* as follows:

$$d(A,B)=\sum_{i=1}^{n}\left|pd(a_i)-pd(b_i)\right|, \qquad (1)$$

where a_i and b_i are the *i*-th items of instances *A* and *B*, respectively. The function *pd*(.) returns the profit density of the given item. *N* is the number of items. (We denote '#items'.)

As the local optimizer to take a local optimum, we use the algorithm as in [9] (see Figure 2). The local optimizer tries to change the weight or value of each item in sequence, and test whether or not the quality of the problem instance increases. If it is true, the change is applied to the instance. Otherwise, the algorithm then considers the next item. It stops after it considers the last item. The changed instance becomes the local optimum for input instance of the local optimizer.

As a representative case, Figure 3 shows the correlation when #items is 20 and the weight coefficient is 0.5. Other correlations are similar to this case. Instances with high *dOthers* and *dBest* have low qualities. But the figures show different shapes from *solution* spaces of well-known combinatorial optimization problems such as graph bi-partitioning (see the shapes in [10]). In [10], when the values of *dBest* are small, the costs of solutions are also small. But Figure 3 shows variant qualities when the values of *dBest* are small. Instances with low *dOthers* and *dBest* could have low qualities. This indicates an obstacle in finding the worst case. On the other hand, there are many instances having the quality 0, which means the best case. This can be another obstacle in finding the worst case, because of the large number of the best cases. Also this result supports the result of [11]. In fact, [11] showed the distribution of this problem instance space is multi-modal.

Table 1 shows numerical data obtained by the fore-mentioned method. We calculate the correlation according to #items and weight coefficient. In our experiments, no local optimum was redundant. The average quality of local optima was quite close to 0. For the fixed #items, the average quality was decreasing as the weight coefficient is increasing. The correlation coefficients are all close to 0. If the coefficients were strong negative values, the search for the worst case would be straightforward.

3 Design of Improved Algorithm

Table 2. Worst-case instances found by the GA of [9] (# of items is 10)

Weight coefficient	Quality	Value sums obtained (Greedy algorithm / optimal one)
0.25	0.960	4 / 101
Instance	(1,1) (1,1) (100,100) (1,35) (1,37) (1,37) (1, 41) (1,46) (1, 51) (1,52) Capacity of knapsack = 101	
0.5	0.925	8 / 106
Instance	(1,1) (1,1) (1,1) (1,1) (1,1) (1,1) (1,1) (100,100) (66,100) (1, 4) Capacity of knapsack = 106	
0.75	0.916	9 / 107
Instance	(1,1) (1,1) (1,1) (1,1) (1,1) (1,1) (1,1) (1,1) (100,100) (1,34) Capacity of knapsack = 107	

*Items in each instance are represented by (*value*, *weight*).

In this section we show the generated test data can be used to find weak points of the tested algorithm and present an improved algorithm. Note that the main points are not the method for finding, but the improved algorithm based on the found. For finding the worst-case problem instance (as defined in Section 2), we use the same method as Jeon and Kim [9] used; a hybrid genetic algorithm (hybrid GA), where the tested algorithm is the greedy algorithm described in Figure 1. Also the local optimizer used in the hybrid GA is the same one as in Figure 2. For the fixed #items and weight coefficient, the method repeats the hybrid GA 50 times independently to achieve the statistical reliability. Jeon and Kim took as the worst case the problem instance showing the highest quality among 50 runs. The results were superior to those obtained by a random testing method.

Table 2 shows some of the worst cases obtained by the fore-mentioned hybrid GA. In Table 2, each item is represented as two-dimensional coordinates (*v*, *w*); *v* is its

value and w is its weight. As an example, suppose that the greedy algorithm tries to solve the worst case in Table 2 when the weight coefficient is 0.5. (Refer to Figure 1 to see how this greedy algorithm works.) In the case, the algorithm takes seven (1,1)s and one (1,4). Then the value sum is 1×7+1×1=8. But six (1,1)s and one (100,100) makes the value sum 1×6+100=106. Since the algorithm prefers lighter items when items have the same profit density, it chooses (1,1) first before considering (100,100). One may suggest the modified algorithm that choose the item with higher value (although it is heavier item) first when considered items have the same profit density. His/her algorithm may overcome the problem instance in Table 2, but it does not perform better in every case than the original algorithm. Consider an instance having one (100,100), one (50,50), two (20,20)s, and four (1,73)s, with the weight coefficient 0.25 (the resulting capacity is 130). Then his/her algorithm will choose only one (100,100) whereas the original algorithm choose one (50,50) and four (20,20)s, which is a better solution. So we suggest a way conducting some post-processing after this algorithm, to perform better in every case than the original algorithm.

The proposed idea is as the following. (The pseudo-code is given in Figure 4.) Once the original greedy algorithm chooses items to be in the knapsack, the post-processing is done from the choice. It takes one of the subsets of the items in the knapsack, and swaps the subset for an item out of the knapsack, in the case that the swapping increases the value sum of the knapsack. Consider again the worst case in Table 2 when the weight coefficient is 0.5. In this case, the post-processing swaps (1,1) and (1,4) in the knapsack for (100,100) out of the knapsack; this achieves the increased value sum 106. This processing gives equal value sum to or higher value sum than one obtained by the original algorithm, at the same time it can overcome some of the worst cases of the original algorithm.

```
U = list of all the given items;
W = given capacity of knapsack;
K = {x∈ U | x is chosen by the original greedy algorithm};
Sort U−K as an array by profit density in non-increasing order†;
for each item_i ∈ U−K
{
  for each set_j ⊂ K
  {
    Δv_j = value of item_i − ∑(value of each item∈ set_j );
    Δw_j = weight of item_i −∑(weight of each item∈ set_j );
  }
  if(∀j, 'Δv_j < 0' or ∑(weight of each item∈K )+Δw_j > W )
    break;
  setB = set_j showing the highest Δv_j ;‡
  Let item_i be in K;
  Let setB be in U−K;
  // we put setB at the highest indices in U−K (unsorted)
}
return K;
```

Figure 4. Pseudo-code for an improved greedy algorithm

†Put lighter item at lower index when the same profit density occurs.

‡When tie occurs, choose set_j showing the lowest Δw_j.

Table 3. Comparison between two algorithms

		Greedy		Greedy II		#samples: 10,000
#items	Weight coefficient	avg Q	sstd Q	avg Q	sstd Q	*p*-value
10	0.25	0.022	0.043	0.004	0.018	4.716E−304
	0.5	0.013	0.025	0.002	0.009	0†
	0.75	0.006	0.015	0.001	0.004	4.219E−273
20	0.25	0.012	0.019	0.003	0.009	0†
	0.5	0.007	0.011	0.002	0.005	0†
	0.75	0.003	0.006	0.001	0.002	0†
30	0.25	0.008	0.011	0.003	0.006	0†
	0.5	0.005	0.006	0.001	0.003	0†
	0.75	0.002	0.004	0.000	0.001	0†

†'0' means that it is quite less than 2.229E−308.

'Q' means quality. 'avg' and 'sstd' mean average and sample standard deviation, respectively. We obtained *p*-values by one-sided *t*-test.

But, considering combinations of the items in the knapsack takes so long time that the new algorithm is too slower than the original algorithm. To reduce the time, we limit the maximum size of the subset being considered to be 2 (i.e., in Figure 4, $|set_j| \leq 2$). Table 3 shows two kinds of the average quality of 10,000 random instances for each classification; one is obtained by taking the original algorithm (we denote 'Greedy') as the tested algorithm and the other one by taking the new algorithm (we denote 'GreedyII'). The low quality means the tested algorithm gives the solution close to the optimal one. The average qualities in the table indicate that the new algorithm is better than the original one.

4 Conclusions

In the approach that defines the performance of algorithms in abstract manner, the algorithm being tested need not to be well-known algorithms; the algorithm might be any part of the software being tested. This paper just illustrates the approach by taking the well-known problem and the well-known algorithm. (Of course, the problem and the algorithm we dealt with are important themselves and thus our experimental results can have some significance in the sense.) For our definition of the quality and given testing algorithm, the problem instance space was somewhat different from well-known solution space and seemed to be difficult to be searched. Also we showed how the test data can be used to devise an improved algorithm (in this paper, a greedy algorithm of the 0/1 knapsack problem); it could overcome some of the worst cases of the original greedy algorithm. Our results will accelerate further researches about the test data generation or empirical analysis of algorithms by generating test data using metaheuristics.

We leave the following studies for future work: (i) finding again the worst case of the improved algorithm and examining the changes of the quality of the worst case, (ii) examining the cost-distance correlation of the improved algorithm, (iii) for the other problems and other definitions of the quality (e.g., one in terms of time complexity), investigating the problem

instance space, (iv) applying different metrics and local optimizers to take the cost-distance correlation and comparing to the original correlation, (v) improving heuristic search algorithm by examining the problem instance space further and redefining the quality so that the landscape becomes more easy-to-search shape, and (vi) finding large-sized worst-case instances from small-sized ones to reduce the search time and the quality evaluation time.

Acknowledgments

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education, Science and Technology(2011-0004215).

5 References

[1] D. Saab, Y. Saab, and J. Abraham. CRIS: A test cultivation program for sequential VLSI circuits. In *Proceedings of 1992 IEEE/ACM International Conference on Computer Aided Design*, pages 216- 219, 1992.

[2] M. Srinvas and L. Patnaik. A simulation-based test generation scheme using genetic algorithms. In *Proceedings International Conference VLSI Design*, pages 132-135, 1993.

[3] E. Rudnick, J. Patel, G. Greenstein, and T. Niermann. Sequential circuit test generation in a genetic algorithm framework. In *Proceedings of the 31st Annual Conference on Design Automation (DAC '94)*, pages 698-704, 1994.

[4] F. Corno, E. Sanchez, M. Sonza Reorda, and G. Squillero. Automatic test program generation - a case study. *IEEE Design & Test*, 21(2):102-109, 2004.

[5] P. McMinn. Search-based Software Test Data Generation: a Survey. *Software Testing Verification And Reliability*, 14(2): 105-156, 2004.

[6] M. Johnson and A. Kosoresow. Finding Worst-Case Instances of, and Lower Bounds for, Online Algorithms Using Genetic Algorithms. *Lecture Notes in Computer Science*, 2557:344-355, 2002.

[7] C. Cotta and P. Moscato. A Mixed-Evolutionary Statistical Analysis of an Algorithm's Complexity. *Applied Mathematics Letters*, 16(1):41-47, 2003.

[8] J. Hemert. Evolving combinatorial problem instances that are difficult to solve. *Evolutionary Computation*, 14(4):433-462, 2006.

[9] S.-Y. Jeon and Y.-H. Kim. A Genetic Approach to Analyze Algorithm Performance Based on the Worst-case Instances. *Journal of Software Engineering and Applications*, 3(8): 767-775, 2010.

[10] Y.-H. Kim and B.-R. Moon. Investigation of the Fitness Landscapes in Graph Bipartitioning: An Empirical Study. *Journal of Heuristics*, 10(2):111-133, 2004.

[11] S.-Y. Jeon and Y.-H. Kim. Finding the Best-case Instances for Analyzing algorithms: Comparing with the Results of Finding the Worst-case Instance. In *Proceedings of the Korea Computer Congress 2010*, vol. 37, no. 2(C), pages 145-150, 2010. (in Korean)

SESSION

SOFTWARE METRICS, PROCESS, AND ALGORITHMS

Chair(s)

TBA

Selection of Software Estimation Models Based on Analysis of Randomization and Spread Parameters in Neural Networks

Cuauhtémoc López-Martín[1], Arturo Chavoya[2], and María Elena Meda-Campaña[3]
[1, 2,3] Information Systems Department, CUCEA, Guadalajara University, Jalisco, Mexico
[1] cuauhtemoc@cucea.udg.mx, [2] achavoya@cucea.udg.mx, [3]emeda@cucea.udg.mx

Abstract - *Neural networks (NN) have demonstrated to be useful for estimating software development effort. A NN can be classified depending of its architecture. A Feedforward neural network (FFNN) and a General Regression Neural Network (GRNN) have two kinds of architectures. A FFNN uses randomization to be trained, whereas a GRNN uses a spread parameter to the same goal. Randomization as well as the spread parameter has influence on the accuracy of the models when they are used for estimating the development effort of software projects. Hence, in this study, an analysis of accuracies is done based on executions of NN involving random numbers and spread values. This study used two separated samples, one of them for training and the other one for validating the models (219 and 132 projects respectively). All projects where developed applying development practices based on Personal Software Process (PSP). Results of this study suggest that an analysis of random and spread parameters should be considered in both training and validation processes for selecting the suitable neural network model.*

Keywords: Software development effort estimation; neural networks, randomization, spread parameter, statistical regression

1 Introduction

An inadequate estimation of the development effort on software projects could address to poor planning, low profitability, and, consequently, products with poor quality [9]. There are several techniques for estimating development effort, which could be classified on intuition-based and model-based. The former is partly based on non-mechanical and unconscious processes and the means of deriving an estimate are not explicit and therefore not repeatable [12]; whereas those ones model-based could be classified on statistical and on computational intelligence techniques. Fuzzy Logic, Genetic Algorithms, Genetic Programming, and Neural networks belong to computational intelligence techniques.

Neural networks have been applied in several fields as accounting, finance, health, medicine, engineering, manufacturing or marketing [10]. According to software development effort estimation, the feedforward is the neural network most commonly used in the effort estimation field [7].

When neural networks have been applied, they have presented the following weaknesses [7]:

1. It is not clear some of the characteristics like sample size or number of variables.
2. The statistical techniques have not been optimally used.
3. Clarity on the determination of parameters of the neural networks.
4. Results obtained from the model building processes are not validated on a new data set that is not used for building the models.

Each of those four problems have considered in this study as follows:

1. Two data samples were used, one of them integrated by 219 projects and developed by 71 persons from the year 2005 to the year 2009, and the other one integrated by 132 projects and developed by 38 persons through the year 2010. Both samples were developed based on the same characteristics of the experiment design (described in section II). Dependent variable is the development effort, whereas independent variables are related to size and people factors, which are described in section I.A.

2. The multiple linear regression equation is generated from a global analysis (based on coefficient of determination) as well as from an individual analysis of its parameters (section IV) to select the significant variables (independent variables) that explain to development effort (dependent variable). This practice has been suggested in [1] and [10].

3. The GRNN contains a parameter named SPREAD which influences in the GRNN accuracy. Accuracy values are analyzed for several SPREAD values (section V). In addition, FFNN involves randomization to be trained, analysis of executions are done in section VI.

4. Analysis of models is based upon the two following main stages when an prediction model is used [4]: (1) the model adequacy checking or model verification (estimation stage) must be determined, that is, whether the model is adequate to describe the observed (actual) data; if so then (2) the estimation model is validated using new data, that is, prediction stage (sections V and VI).

Data of this study were obtained by means of the application of a disciplined software development process: the Personal

Software Process (PSP) whose practices and methods have been used by thousands of software engineers for delivering quality products on predictable schedule [5].

1.1 Data description of software projects

Source lines of code (LOC) remains in favor of many models [14]. There are two measures of source code size: physical source lines and logical source statements. The count of physical lines gives the size in terms of the physical length of the code as it appears when printed [11].

In this study, two of the independent variables are New and Changed (N&C) as well as Reused code and all of them were considered as physical lines of code (LOC). N&C is composed of added and modified code. The added code is the LOC written during the current programming process, while the modified code is the LOC changed in the base program when modifying a previously developed program. The base program is the total LOC of the previous project while the reused code is the LOC of previously developed programs that are used without any modification.

A coding standard should establish a consistent set of coding practices that is used as a criterion when judging the quality of the produced code. Hence, it is necessary to always use the same coding and counting standards. The software projects of this study followed those two guidelines.

After product size, people factors (such as experience on applications), platforms, languages and tools have the strongest influence in determining the amount of effort required to develop a software product [2]. Programming language experience is used as a third independent variable in this study, which was measured in months. Because projects of this study were developed inside an academic environment, the effort was measured in minutes as was used in [16].

1.2 Accuracy criterion

There are several criteria to evaluate the accuracy of estimation models. A common criterion for the evaluation of prediction models has been the Magnitude of Relative Error (MRE). In several papers, a MMRE≤0.25 has been considered as acceptable.

The accuracy criterion for evaluating models of this study is the Magnitude of Error Relative to the estimate or MER defined as follows:

$$MER_i = \frac{|\text{Actual Effort}_i - \text{Estimated Effort}_i|}{\text{Estimated Effort}_i}$$

The MER value is calculated for each observation *i* whose effort is estimated. The aggregation of MER over multiple observations (N) can be achieved through the mean (MMER) as follows:

$$\text{MMER} = (1/N)\sum_{i=1}^{N} MER_i$$

The accuracy of an estimation technique is inversely proportional to the MMER.

Results of MMER had better results than MMRE in in [15] for selecting the best model; this fact is the reason for using MMER

2 Experimental design

The experiment was done inside a controlled environment having the following characteristics:

1. All of the developers were experienced working for software development inside of their enterprises which they were working.
2. All developers were studying a postgraduate program related to computer science.
3. Each developer wrote seven project assignments. However only four of them were selected by developer. The first three programs were not considered because they had differences in their process phases and in their logs, whereas in latest four programs were based on the same logs and in the following phases: plan, design, design review, code, code review, compile, testing and postmortem.
4. Each developer selected his/her own imperative programming language whose code standard had the following characteristics: each compiler directive, variable declaration, constant definition, delimiter, assign sentence, as well as flow control statement was written in a line of code.
5. Developers had already received at least a formal course about the object oriented programming language that they selected to be used though the assignments, and they had good programming experience in that language. Sample of this study only involved developers whose programs were coded in C++ or JAVA.
6. Because of this study was an experiment with the aim to reduce bias, we did not inform to developers our experimental goal.
7. Developers fill out an spreadsheet for each task and submit it electronically for examination.
8. Each group course was not greater than fifteen developers.
9. Since that a coding standard should establish a consistent set of coding practices that is used as a criterion when judging the quality of the produced code [16], it is necessary to always use the same coding and counting standards. The programs developed of this study followed these guidelines. All of them coincided with the counting standard depicted in Table I.
10. Developers were constantly supervised and advising about the process.
11. The code wrote in each program was designed by the developers to be reused in next programs.
12. The kind of the developed programs had a similar complexity of those suggested in [16].
13. Data used in this study belong from those, whose data for all seven exercises were correct, complete, and consistent

Table 1. Counting standard

Count type	**Type**
Physical/logical	Physical
Statement type	**Included**
Executable	Yes
Nonexecutable	
Declarations	Yes (one by text line)
Compiler directives	Yes (one by text line)
Comments and Blank lines	No
Delimiters:	
{ and *}*	Yes

3 Neural networks

An artificial neural network, or simply a neural network (NN), is a technique of computing and signal processing that is inspired on the processing done by a network of biological neurons [13]. The basis for construction of a neural network is an artificial neuron. An artificial neuron implements a mathematical model of a biological neuron.

There is a variety of tasks that neural network can be trained to perform. The most common tasks are pattern association, pattern recognition, function approximation, automatic control, filtering and beam-forming.
The neuron model and the architecture of a neural network describe how a network transforms its input into an output. Two or more neurons can be combined in a layer, and a particular network could contain one or more such layers [6]. Two kinds of neural networks are briefly described in the following two sections

3.1 Feedforward neural network (FFNN)

The input to an artificial neuron is a vector of numeric values $\vec{x} = \{x_1, x_2, ..., x_j, ..., x_m\}$. The neuron receives the vector and perceives each value, or component of the vector, with a particular independent sensitivity called weight $\vec{w} = \{w_1, w_2, ..., w_j, ..., w_m\}$. Upon receiving the input vector, the neuron first calculates its internal state *v*, and then its output value *y*. The internal state *v* of the neuron is calculated as the sum of the inner product of the input vector and the weight vector, and a numerical value *b* called "bias" as follows: $v = \vec{x} \cdot \vec{w} + b = \sum_{j=1}^{m} y_j w_j + b$. This function is also known as "transfer function". The output of the neuron is a function of its internal state $y = \Phi(v)$. This function is also known as "activation function". The main task of the activation function is to scale all possible values of the internal state into a desired interval of output values. The intervals of output values are for instance [0, 1] or (-1, 1). A feedforward network consists of layers of neurons. There is an input layer, an output layer and optionally one or more hidden layers between the input and the output layers. After a network receives its input vector, layer by layer of neurons process the signal, until the output layer emits an output vector as response. Neurons in the same layer process the signal in parallel. In the feedforward network the signals between neurons always flow from the input layer toward the output layer.

A neural network learns by adjusting its parameters. The parameters are the values of bias and weights in its neurons. Some neural networks learn constantly during their application, while most of them have two distinct periods: training period and application period. During the training period a network processes inputs adjusting its parameters. It is guided by some learning algorithm, in order to improve its performance. Once the performance is acceptably accurate, or precise, the training period is completed. The parameters of the network are then fixed to the learned values, and the network starts its period of application for the intended task.
In the present work, a feedforward neural network with one hidden layer is applied for function approximation.

3.2 General regression neural network (GRNN)

The architecture of a GRNN is the following [3]: input units provide all the X_i variables to all neurons on the second layer. Pattern units are dedicated to receive as input the outputs from a set of input neurons. When a new vector X is entered into the network, it is subtracted from the stored vector representing each cluster center. Either the squares or the absolute values of the differences are summed and fed into a nonlinear activation function. The activation function normally used is the exponential function. The pattern units' output is passed on to the summation units. The summation units perform a dot product between a weight vector and a vector composed of the signals from the pattern units. The summation unit that generates an estimate of *F(X)K* sums the outputs of the pattern units weighted by the number of observations each cluster center represents. The summation unit that estimates *Y´ F(X)K* multiplies each value from a pattern unit by the sum of the samples Y^j associated with cluster center X^i. The output unit merely divides *Y´ f(X)K* by *f(X)K* to yield the desired estimate of *Y*. When estimation of a vector *Y* is desired, each component is estimated using one extra summation unit, which uses as its multipliers sums of samples of the component of *Y* associated with each cluster center X_i.

4 Significant variables from statistical regression analysis

From a sample of 219 projects, the following multiple linear regression equation considering New and Changed

(N&C), Reused code and Programming Language Experience (PLE) was generated:

$$\text{Effort} = 62.5307 + (1.1025*\text{N\&C}) - (0.189257*\text{Reused}) - (0.477072*\text{PLE})$$

This equation has a coefficient of determination of $r^2 \geq 0.51$, which corresponds to an acceptable value in software estimation according to [16]. ANOVA for this equation showed a statistically significant relationship between the variables at the 99% confidence level. To determine whether the model could be simplified, a parameters analysis of the multiple linear regression was done. In results for this analysis (Table 2); the highest p-value on the independent variables was 0.0027, belonging to reused code. Since this p-value was less than 0.05, reused code is statistically significant at the 95% confidence level. Consequently, the independent variable of reused code was not removed. Hence, this variable will have to be considered for its evaluation.

Table 2. Individual analysis of parameters

Parameter	Estimate	Standard error	t-statistic	p-value
Constant	62.5307	4.6836	13.3509	0.0000
N&C	1.1025	0.0766	14.3819	0.0000
Reused	-0.189257	0.0623	-3.0361	0.0027
PLE	-0.477072	0.1028	-4.6364	0.0000

5 Analysis of GRNN spread parameter

In GRNN a parameter named SPREAD was empirically changed until a suitable value was obtained. If the parameter spread is small the GRNN function is very steep, so that the neuron with the weight vector closest to the input will have a much larger output than other neurons. The GRNN tends to respond with the target vector associated with the nearest input vector. As the parameter spread becomes larger, the function slope of the GRNN becomes smoother and several neurons can respond to an input vector. The network then acts as if it is taking a weighted average between target vectors whose input vectors are closest to the new input vector. As the parameter spread becomes larger, more and more neurons contribute to the average, with the result that the network function becomes smoother [6]. The values for SPREAD were 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20, 25, 30, 35, and 40. To select suitable GRNN based on its spread value, it is necessary to know the behavior when the GRNN is applied to a new dataset; that is, a low spread value could over-fit the network and then when the GRNN is applied to new data, it could obtain a larger (worse) MMER instead of a better one. Table 3 shows the MMER values in both the verification and the validation stages as the spread value is being increased. Table 3 shows that as the spread value increases, the MMER in validation stage gets better until the spread value is equal to 13 (when MMER has its best value with 0.23). It can be observed that from the spread value equal to 25, the MMER gets worse. Hence, we considered as suitable GRNN that having a spread value equal to 13.

Table 3. Analysis of MMER by stage based on GRNN spread value

SPREAD values	Stage	
	Verification (219 projects)	Validation (132 projects)
5	0.14	0.28
6	0.16	0.26
7	0.18	0.25
8	0.20	0.25
9	0.21	0.24
10	0.22	0.24
11	0.23	0.24
12	0.23	0.26
13	0.24	0.23
14	0.24	0.23
15	0.25	0.23
20	0.26	0.23
25	0.28	0.24
30	0.29	0.24
35	0.29	0.25
40	0.30	0.26

6 Analysis of FFNN randomization

A feedforward network with one layer of hidden neurons is sufficient to approximate any function with finite number of discontinuities on any given interval [13]. Three neurons were used in the input layer of the network. One receives a number of N&C, the second one receives the number of reused lines of code, whereas the last one receives the developer's programming language experience in months. The output layer consists of only one neuron indicating an estimated effort. The set of 219 software projects was used to train the network. This group of projects was randomly separated in three subgroups: training, validation and testing. The training group contained 60% of the projects. The input-output pairs of data for these projects were used by the network to adjust its parameters. The next 20% of data were used to validate the results and identify the point at which the training should stop. The remaining 20% of data were randomly chosen to be used as testing data, to make sure that the network performed well with the data that was not present during parameter adjustment. These percentages were chosen as suggested in [32].

The number of neurons in the hidden layer was optimized empirically: 1, 2, 3, 4, 5, 10, 15, 20, 25 and 30 neurons were used for training the network. Ten executions were done by each number of neurons because this kind of network involved a random process. The optimized Levenberg-Marquardt algorithm [8] was used to train the network. Table 4 presents the MMER obtained by execution.

Table 4. MMER by execution having different number of neurons in the hidden layer

Neurons by hidden layer	Executions										Best MMER
	1	2	3	4	5	6	7	8	9	10	
1	0.26	0.25	0.25	0.26	0.25	0.27	0.25	0.25	0.26	0.25	0.25
2	0.24	0.25	0.25	0.26	0.26	0.26	0.26	0.26	0.26	0.25	0.24
3	0.25	0.24	0.25	0.25	0.25	0.28	0.25	0.28	0.25	0.25	0.24
4	0.27	0.26	0.25	0.24	0.26	0.25	0.25	0.25	0.25	0.25	0.24
5	0.26	0.25	0.26	0.26	0.25	0.24	0.25	0.24	0.26	0.29	0.24
10	0.25	0.26	0.27	0.25	0.24	0.25	0.24	0.25	0.28	0.26	0.24
15	0.24	0.38	0.25	0.26	0.25	0.25	0.25	0.25	0.27	0.24	0.24
20	0.25	0.25	0.25	0.25	0.24	0.32	0.24	0.26	0.25	0.25	0.24
25	0.33	0.28	0.38	0.36	0.24	0.28	0.30	0.25	0.25	0.24	0.24
30	0.26	0.24	0.33	0.42	0.25	0.24	0.35	0.28	0.24	0.25	0.24

Table 4 shows a MMER = 0.24 using from 2 to 30 neurons. Considering that more neurons means more computation, to select the final number neurons to be used in this study we proceeded to analyze the frequency of MMER. Table 5 shows that the higher the number of neurons, the higher the MMER dispersion.

Table 6. Frequency of MMER by number of neurons in the hidden layer

MMER	Number of neurons in the hidden layer									
	1	2	3	4	5	10	15	20	25	30
0.24		1	1	1	2	2	2	2	2	3
0.25	6	3	7	6	3	4	5	6	2	2
0.26	3	6		2	4	2	1	1		1
0.27	1			1		1	1			
0.28			2			1			2	1
0.29					1					
0.30									1	
0.32								1		
0.33									1	1
0.35										1
0.36									1	
0.38							1		1	
0.42										1
Total of executions	10	10	10	10	10	10	10	10	10	10

Based on the data from Table 6, we selected as suitable neural network that with three neurons in the hidden layer since it had the highest frequency (seven times) having a MMER = 0.25. Then, this trained FFNN was applied to the other data set of 132 projects obtaining a MMER = 0.24.

7 Conclusions and future research

This research has analyzed the effect that randomization and spread parameter have on the selection of the best neural network model. Accuracy was measured based on the Mean of Magnitude of Error Relative to the estimate or MMER. Two kinds of neural networks were analyzed. The randomization involved in a FFNN showed that the higher the number of neurons, the higher the MMER dispersion. In accordance with GRNN spread parameter, our analysis showed that to select suitable GRNN, it is necessary to know the behavior when the GRNN is applied to a new dataset and not only is sufficient to know its accuracy of the GRNN when it is trained. This analysis was inside of a software development estimation context based upon projects developed in a controlled environment as well as following a disciplined process. Future work is related to the relationship analysis between data statistical characteristics and accuracy of estimation models

8 Acknowledgement

The authors of this paper would like to thank CUCEA of Guadalajara University, Jalisco, México, Programa de Mejoramiento del Profesorado (PROMEP), as well as to Consejo Nacional de Ciencia y Tecnología (Conacyt).

9 References

[1] B. A. Kitchenham and E. Mendes, "Travassos G.H. (2007). Cross versus Within-Company Cost Estimation Studies: A Systematic Review", IEEE Transactions Software Engineering", Vol. 33, No. 5, pages, 316-329

[2] B. Boehm Ch. Abts, A.W. Brown, S. Chulani, B.K. Clarck, E. Horowitz, R. Madachy, D. Reifer and B. Steece, 2000, COCOMO II. Prentice Hall.

[3] D. F. Specht, "A General Regression Neural Network. IEEE transactions on Neural Networks", Vol. 7, No. 3, 1991.

[4] D. Montgomery and E. Peck, "Introduction to Linear Regression Analysis, 2001, John Wiley.

[5] D. Rombach, J. Münch, A. Ocampo, W. S. Humphrey and D. Burton, "Teaching disciplined software development. Journal Systems and Software", Elsevier, 2008, pp. 747- 763.

[6] H. Demuth, M. Beale and M Hagan, MatLab Neural Network Toolbox 6, User's Guide, 2008.

[7] H. Park and S. Baek, "An empirical validation of a neural network model for software effort estimation", Journal of Expert Systems with Applications, Elsevier, 2008, Vol. 35, Pp. 929–937

[8] L. Finschi, "An Implementation of The Levenberg-Marquardt Algorithm". Eidgenössische Technische Hochschule Zürich, 1996

[9] M. Jørgensen, "A Preliminary Theory of Judgment-based Project Software Effort Predictions". IRNOP VIII, Project Research Conference, ed. by Lixiong Ou, Rodney Turner, Beijing, Publishing House of Electronic Industry, 2006, pp. 661-668

[10] M. Paliwal and U.A.Kumar, "Neural networks and statistical techniques: A review of applications", Journal of Expert Systems with Applications, Vol. 36, Pp.2–17. 2009. doi:10.1016/j.eswa.2007.10.005

[11] R.E. Park. "Software Size Measurement: A Framework for Counting Source Statements", 1992. Software Engineering Institute, Carnegie Mellon University.

[12] S. Grimstad and M. Jørgensen, "Inconsistency of expert judgment-based estimates of software development effort",

Journal of Systems and Software**,** Elsevier, Vol. 80, 2007 pp. 1770–1777.

[13] S. Haykin, "Neural Networks: A Comprehensive Foundation", Second edition, Prentice Hall. 1998

[14] S.G. MacDonell, "Software source code sizing using fuzzy logic modelling", Elsevier. Volume 45, Issue 7, 2003, pp. 389-404. Doi:10.1016/S0950-5849(03)00011-9

[15] T. Foss, E. Stensrud, B. Kitchenham and I. Myrtveit I, "A Simulation Study of the Model Evaluation Criterion MMRE", IEEE Transactions on Software Engineering, 2003, Vol. 29, No. 11.

[16] W. Humphrey "A Discipline for Software Engineering". Addison Wesley. 1995.

Effects of Culture on Software Project Team Management

Mohammad A. Alkandari[1], **and Shawn A. Bohner**[2]
[1]Department of Computer Science, Virginia Tech, Blacksburg, VA, USA
[2]Department of Computer Science and Software Engineering, Rose-Hulman Institute of Technology, Terre Haute, IN, USA

Abstract - *In today's world of global software teams, managing members from multiple countries and cultures adds to an already complex mix of project factors. Based on a survey of multicultural teams, this paper examines some key issues that arise on culturally diverse teams. We focused on communications and conflict resolution factors, and their respective impact on productivity. Results of this led to language and attitudes as dominant elements of communications that impact productivity. Time, roles, and social organization also had considerable influence. We formulated a research approach that examines these elements and designed more specific survey instruments to aid in teasing out the critical factors that impact communication in multicultural software teams. Initial results of the deeper surveys show that various cultures have different attitudes, which in turn, have distinct impacts on communication. Based on this research, we are continuing to further define and refine the Multicultural Software Project Team model.*

Keywords: Team Management, Software Project, Culture, Productivity, Requirements Engineering (RE).

1 Introduction

This research focuses on managing software project team members from different cultures – an increasingly reality in today's global economy [7]. The objective of our research is to investigate how individuals from different cultures communicate and work effectively together in software development projects. With this understanding, the intent is to formulate a model from which to reason about key cultural factors (e.g., mitigating risks and reducing conflicts by reducing the amount of miscommunications that might occur in multicultural teams, reducing reworks and delays, understanding and solving problems). This research shows how cultural factors (e.g. attitudes, language, time, roles, and social organization) [6] might affect the communication process in software development projects, which in turn affect the overall software project management and productivity.

Not all software teams are created equal from a productivity perspective. Software team productivity has notionally been associated with team member skills, roles, experience, and the like. Team cohesiveness and communication are also keys. Multicultural teams often affect these key factors.

Software productivity in its simplest form is product output divided by the input of resources (e.g., lines of code per staff month) [2]. For example, if the product output has defects and some percent needs to be discarded or reworked, this diminishes the productivity. Hence, rework and delays are considered key productivity factors. When errors are introduced through miscommunication (e.g., requirements misunderstood), rework increases the time and effort necessary to complete the project.

Software team management models based on personality, and roles [15, 8, 1] serve as the basis for our work examining cultural issues in software project teams. The configurations of the team in terms of personalities and roles have been shown to have an impact on the software team's performance. With today's software being developed by multi-cultural team, we must understand the impact that culture has on software teams. Recognizing that much of the critical communication occurs in the beginning, we examine the effect of cultural on requirements engineering.

John states eloquently, "human and social factors have a very strong impact on the success of software development endeavors and the resulting system [12]." Many cultural challenges exist in global software development and specifically in requirements engineering [10, 3]. A model is needed to consider the social aspects in order to better manage multicultural software teams [11]. Hence, this research investigates the impact of cultural factors on software development teams with respect to team productivity.

An informal study was conducted as a first step towards bounding and focusing this research. The informal study was an interview-based survey where software engineers and experts from multicultural projects were questioned to help understand the implications of culture in software development teams. The survey was designed to glean a preliminary understanding of how cultural factors are perceived by the software engineering community and to help scope the research towards the most relevant factors. The interview was designed to help formulate our first set of questions (questionnaires) that were used in the formal studies. This paper reports on what has been captured and analyzed thus far, outlining preliminary conclusions.

2 Motivation

While software team compositions have been researched based on tasks, personality, and role descriptions, few if any, models exist to reason about teams with respect to culture. As more culturally diverse teams develop software products, project managers need to reason about how to manage teams based on cultural factors that will inevitably arise. Software engineering is a team endeavor that entails using an expensive resource that is often highly complex and variable – people [14]. In general, people vary according to their cognitive and personality styles; behaviors; experiences; abilities; education and training; motivation; management; and communications [13, 7]. Each software engineering role involves a personality type [15, 9] and each team member belongs to a culture, which, in turn, has its own characteristics. Different cultures often think and communicate differently, and teams with these varied cultures have different approaches to solving problems. Hence, the "people factor" is so important that software engineers should take into consideration how team members can work effectively together. Gifford expressed the need for research investigating individual differences in personalities with respect to their cultures as a source for helping software managers structure their teams [8]. People have influence on software productivity and its impact intensifies as software systems expand in size and complexity [5]. Communication is an important process in the requirements engineering phase; not only the ones that take place between a customer and a software engineer, but also communications between team members.

3 Culture in Team Management

On multicultural teams, you may not be able to select the team members, but the management techniques you employ can certainly be adjusted according to the personalities and cultures of the team members. Understanding the cultural profiles of individuals in software teams would positively affect the project's outcome by avoiding conflicts, miscommunications, and other potential situations.

Deresky [6] outlines seven cultural variables that independently influence the communication process: The factors are discussed here to give an appreciation for the complexities that can arise in software projects.

1) Attitudes: determines the way people think, feel, behave and interpret messages from others.

2) Social Organization: describes the way that values, methods, and priorities of a particular social organization could affect people's behaviors.

3) Thought Patterns: the influence of variations in the logical progression of reasoning between cultures.

4) Roles: specify the ways people recognize manager as those who make decisions and assign responsibilities.

5) Language: specifies the communication language used.

6) Nonverbal communication (body language): describes the behavior that causes interaction and communication, but without words.

7) Time: identifies the different ways that people value, treat, and use time.

Each of these plays a role in the way people from different cultures view communications, and are expressed in our research model.

4 Problem Statement

Very few software engineering studies over the past decade have focused on the human side of software engineering – rather, they have predominantly concentrated on the analysis, design, and construction techniques necessary to produce software [15, 8, 12]. There has been some research outside of the software engineering domain that aids teambuilding, promotes teamwork, and fosters communication, coordination, and collaboration. A small number of studies have addressed the personality composition of team members within a software project context [9]. Some addressed the importance of studying the human aspects of software engineering with respect to personalities and cultural identities and investigate its influence on the software development life cycle as well as find out its relationship with productivity issues [4, 5]. However, we found no work presenting the relationship between personality profiles and cultural differences, or a complete model of software teams for mitigating risks and managing conflicts based on the cultural characteristics of team members. In other words, no work has addressed the effects of the cultural factors on the communication process in multicultural software teams.

Problem: Software Project Teams increasingly consist of members from multiple cultures. Yet for software project teams, little if any research exists on cultural differences, and the communication and productivity implications.

What is needed is a model of software project teams that allows us to reason about culture factors in software team productivity. This would help software managers understand how team members in software development projects from different cultures perform high quality software work. It would assist project managers to coordinate team members more effectively to achieve successful communication and interaction in multicultural teams. This should help project managers combat the complexity, confusion, and other significant difficulties the multicultural software teams encounter while accomplishing their project tasks.

The first objective of this research is to understand those aspects of culture that influence the communication process and dominate software team productivity in terms of rework and delay.

Studies show that more software products are being developed by multicultural teams [3, 5, 10]. Diverse culture often has a negative impact on producing software. Thus, the main *objective* of this research investigates specific team communication cultural variables, and their potential impact on software productivity. That is, the research intent is to explore how cultural factors impact software team communication and subsequently software rework and delay. Therefore, determining the key cultural aspects that lead to

less productivity in terms of rework and delay is one of the major questions/concerns of this research study.

The *goals* of this research are:

1) To understand conflicts that might exist in software teams, and investigate if any are related directly or indirectly to cultural factors.

2) For Software Project Managers to better reason about their multicultural teams, a framework must be developed that addresses communication and conflict resolution in order to improve software productivity.

5 Research Approach

Our research approach is to identify key factors from (e.g., Deresky's work with international management teams [6]) to formulate multicultural software team model. As stated earlier, this is initially done through an informal exploratory survey designed to identify these factors and presented to several multicultural software teams. Based on the informal study's results, a more in-depth study is designed to drill down on key elements to refine understanding and demonstrate alignment with expert opinion. Then, we focus on the top three to four factors with more depth as well as iterate and refine as the body of knowledge dictates. Based on the results of the informal and formal studies, a model for the Multicultural Software Project Team will be developed to reason about key cultural factors.

For purposes of this study, the scope was limited to software projects, and specifically focused on communication software requirements engineering tasks. The context reflects impacts of cultural factors on software productivity.

In the next two sections, we present the informal and formal studies with their respective findings. We then discuss the implications for software project teams and outline the future research to be completed this year.

6 Informal Study

The informal study is a survey/interview-based study designed to help discern the implications of culture in software development teams. The survey was used to glean a preliminary understanding of how cultural factors are perceived by the software engineering community in order to establish the basis for our future research instruments. The interview aided in formulating our first set of questions used in the formal studies.

The interviews were prepared for software engineers who have experience in software development projects working either in multi-cultural teams, same-cultural teams, or both. 18 participants were asked to fill out the survey and answer a set of discussion questions (total of 13 questions), looking for their background circumstances or their experiences/knowledge in software development projects (insights and opinions). The provided information will be used to improve the process of conducting the subsequent studies as well as improve the design of the material that is used in this research.

More specifically, the objective of the informal study was to get an indication on: which cultural factors (i.e., thought patterns, language, social organization, time, roles.) might have the most impact on team's communication and under what conditions. This is to identify how differences in cultural factors might affect communication in requirements engineering, and in turn the amount of rework and delay.

6.1 Informal Study's Analysis and Results

The informal study's survey consists of two sections: General Questions and Specific Questions. The first section asks some general questions (e.g. time/date and place of interview, survey facilitator's name and participant's name, years of experience and number of projects, as well as level of education). In this study, 18 software engineers were interviewed providing their knowledge/experience working in multicultural software teams. The average and the median of the participants' years of experience in the field of software engineering are around 8.5 years. The average of the number of projects the participants got involved in is around 24 projects while the median is around 17.5 projects. All 18 participants have experience working in both multicultural teams and uni-cultural teams and the majority of the participants have graduate degrees, in which all of them were qualified to participate in this study.

The second set of the questions addresses specific information about their knowledge, experience, and opinions with respect to software engineering development projects, and software teams. The participants were given a short statement about communication in software teams and discussed the cultural factors that might influence the communication process. Based on Deresky's model, seven cultural factors might influence communication in multi-cultural teams: Social organization (values, methods, priorities), attitudes (interpret messages), language (express feelings, translate idioms), thought patterns (reasoning process), roles, time, nonverbal communication. Therefore, the participants were asked to answer two sets of questions where the first one focuses on their experience on identifying which of these culture factors they believe have the most dominant impact on software teams they have been involved, while the second one focuses on determining which of these factors might have a high effect on the communication process in software teams and requirements engineering in particular. For this question, each cultural factor was explained in detail with some scenarios that explain each cultural aspect by definition and within the software engineering context as well. The main purpose of discussing these cultural factors was not only to give examples for easy understanding but also to ensure that the participants were answering the questions based on the cultural variables not the differences in personalities.

The participants were asked to make their choices on a Likert scale from 1 to 5, where (1) represents no impact and (5) represents catastrophic impact. Based on the literature review, all the seven factors might have impact on multicultural teams and some of them might overlap based on some common characteristics. However, for this study, the

aim is to narrow down the focus to 3-4 factors instead of investigating all of the seven factors in the formal studies due to limited research time frame and to make this research more focused on specific cultural aspects. To this end, the resulting numbers shown in Figure 1 reflect the fact that all of the factors show some level of importance ranging from 2 to 4.

Attitude and language indicated the impact on both the overall software team management and the communication process in requirements engineering (RE) phase. The average of the attitude is approximately 3.7 with a median of 4 for the first part of the question where the focus was on the overall team management and around 3.6 with a median of 4 for the second part of the question where the focus was on the communication process in the RE. The average of the language is around 3.44 with a median of 3, which has nominal impact for the first part. However, the average of the language is just about 3.7 with a median of 4, which has more impact on the communication process in RE.

Time, roles, and social organization appeared less critical than attitude and language, but at the same time, considered important as secondary factors to be focused on, and involved in the study since some of these factors might overlap with language and attitude and share some of its characteristics. The average of time is around 3.33 with a median of 3.5 which means it is somewhere between having nominal impact and more impact on the overall software project. Moreover, the average of time is just about 2.9 with a median of 3, which means it is much closer to have nominal impact on communication process that takes place in the RE. The average of social organization is around 3.2 with a median of 3 for both parts, which means it has same nominal effect on the overall software team management and team communication. The average of roles is approximately 3.3 with a median of 3 for the first part of the question and an average of 2.8 with a median of 3 for the second part of the question. Thus, it is between having less impact and nominal impact, but closer to the normal influence.

Therefore, the differences between time, roles, and social organization are somehow negligible and these factors might overlap and share some features with language and attitude. Thus, these factors may need further investigation in subsequent formal studies.

Thought patterns and nonverbal communication appeared less dominant. Therefore, they were focused on our next study. This does not mean that these two factors are eliminated, but the formal studies will focus on the first five factors due to the limited time frame of this research.

As shown in Figure 1, the average of thought patterns is around 2.4 with a median of 2.5 for the first part of the question, and an average of 2.7 with a median of 2.5 for its impact on the communication in RE. The average of nonverbal communication is approximately 2.5 with a median of 2 for its impact on the overall team management, and an average of 2.7 with a median of 3 for its impact on the communication in RE. Thus, both thought patterns and nonverbal communication have less impact, which is considered negligible and very manageable.

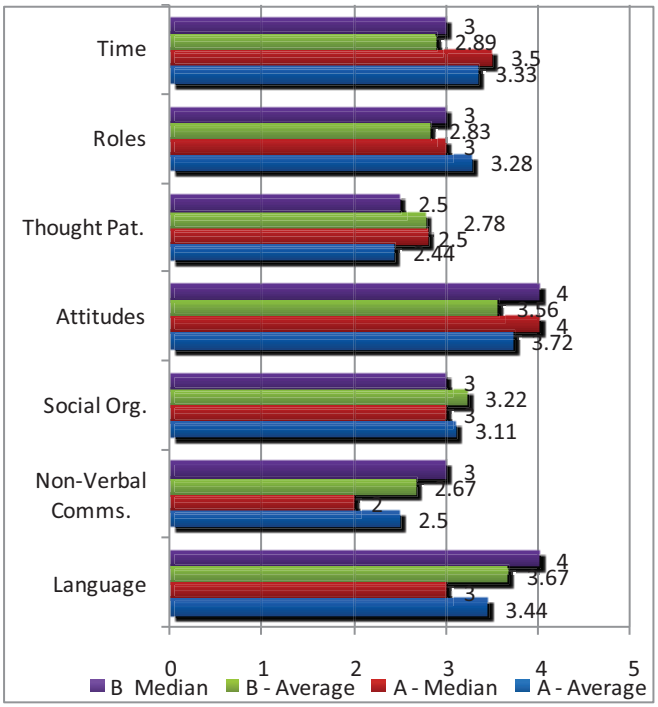


Fig. 1: Average & median cultural factors

The first set of cultural factors (attitude and language) appears to be the ones that have the impact on both the overall software team management and communication in RE. The second set of cultural factors (time, roles, and social organization) influence the project enough that they warrant attention too. However, the last two factors (thought patterns, and nonverbal communication) appear to be less important based on the data; however, they might be studied and investigated further in the future due to limited time frame. Figure 2 illustrates the focus on attitudes and language factors, followed by time, roles, and social organization.

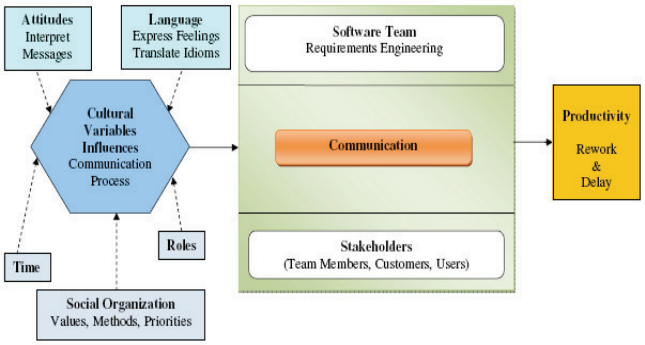


Fig. 2: Focus of the Research Approach

The third part of the survey, involves more discussion questions to better understand the participants' implications and prospective with respect to cultural aspects in software development teams. 17 out of 18 participants supported the argument that cultural differences might cause miscommunication, which in turn, might cause some conflicts resulting in errors and reducing productivity in terms of

rework and delay. They provided some opinions regarding their experiences working in multicultural software teams, where some of them believe that cultural differences are a big challenge for someone who is new to multicultural teams. They supported the base of the research where culture is becoming an important topic to be discussed nowadays in software project management. These participants stated that miscommunication occurs often in software teams and some of them they see it clearly related to diversity in cultures. For example, some participants noted that misunderstandings happen frequently, but most of the time they are resolved during reviews. Yet, those reviews consume a lot of time and effort, especially in large-scale projects, which often cause delay and rework in the project. Others indicated that social values have negative impact on software projects since it causes a lot of miscommunication and misunderstanding.

17 out of 18 participants supported the argument where culture may affect requirements engineering (RE) activities in many situations, especially when eliciting and negotiating requirements. They stated that cultural differences may cause misunderstanding in expressing/explaining the requirements which in turn, may lead to wrong requirements specification. Moreover, some of them mentioned that different cultures have unique prospective on problems and how to address them in which, some people want to discuss everything upfront to have a perfect design while others take a more iterative approach. Additionally, based on their experiences, cultural factors influence software teams where for example, some cultures tend to be less willing to argue or create conflict. This could lead to group members agreeing with decisions they do not actually support. In other words, some cultures just want to be agreeable which might cause conflicts in future.

Some participants discussed how it is difficult to deal with stakeholders globally since it requires understanding how to communicate with them in various circumstances to elicit their requirements. This requires a lot of communication, informal talks, and meetings, which sometimes cause delay and rework. For example, one of the participants discussed the importance of understanding the stakeholders' cultures in order to get the right requirements, in which they used to spend more time before the meetings discussing the customer cultural aspects to make it easier understanding his/her needs. They sometimes have a problem interpreting messages and understanding their needs.

The majority of the participants supported conducting this study on people who are engineering the requirements since this is a communications intensive part of the life cycle process that drives much of the subsequent software activities. Seven participants indicated that other phases in the software development life cycle might be better to focus on in future work. Other phases are: testing, delivery/deployment, maintenance, reviews, quality control, design and architecture, implementation, and modifications sessions.

11 out of 18 participants think that language and attitude are the major factors that might affect requirements engineering and pair-programming as well. They stated that differences in Languages (Terminologies) lead to miscommunication. Differences in language (spoken/written) can cause misunderstanding because of misinterpretation, which in turn, wastes time. Differences in accents and phrasing lead to misunderstanding as well. A participant described language as an important factor that might cause considerable conflicts, which require more meetings to understand and resolve. As a workaround, some of the participants sent email after several meetings to clarify what they meant. The problem was resolved, but caused delays. Language also causes misunderstanding in requirements' elicitation (e.g. meant something different every time), which required more meetings that caused delays. Based on their experiences, interpretation and understanding the implicit message is the problem in multicultural teams. Another stated experience indicated that working with people from different cultures consumes a lot of time to understand what they say. Communication is an important factor that has huge productivity implications based on experience working in a private company. It is really clear that language is one of the critical factors that affect the communication process in RE, and in turn the productivity.

7 Formal Studies

The intent of the formal studies is to produce a model from which software project managers can reason about their multicultural teams. Using this model, it is hoped that project managers can increase the level of communication and reduce culture related conflicts.

Two studies were designed and prepared, in which the first one was conducted on software engineers who have experience in the field of software engineering and software development projects working either in multi-cultural teams or same-cultural teams or both, while the second one was conducted on software engineers in industry, who are working on a particular software development project in a multi-cultural team or same-cultural team. Therefore, two research instruments (questionnaires) have been designed for the formal studies aiming to investigate the impact of cultural variables on software project teams and productivity. The questionnaires generally consist of three main parts: the first part includes questions that measure each cultural factor (e.g. questions to measure attitudes, language, and so on).

The second part consists of questions that measure communication, and the third part includes questions that relate the communication to the productivity in a causality manner. Noting that the surveys are built based on previous validated instruments for measuring the cultural factors and communication in teams. Currently, we are in the process of conducting the research formal studies and collecting data. However, we have started analyzing the first set of data that resulted from both formal studies, and got some initial contributions.

7.1 Formal Studies' Initial Contributions

The first formal study was conducted on software engineers who have experience in the field of software

engineering. Participants worked on software development projects in multi-cultural teams, same-cultural teams or both. Forty-eight people participated so far in this study, where 85% of them have experience in multi-cultural teams, 88% have experience in same-cultural teams Sixty-two percent of the participants have more than four years experience working in multi-cultural software teams, and 50% of them have worked on more than 10 projects.

Based on the initial analysis of the data, the study shows that there is a difference between how people communicate with teammates in multicultural teams and how they interact in same-cultural teams. The study shows that people sometimes have difficulties expressing ideas, and translating idioms, and sometimes fail to convey messages to others in multicultural teams. The study also indicates that people in multicultural teams sometimes missed the meaning of the messages conveyed by others and have misunderstandings where they sometimes feel unable to communicate what they really mean. Moreover, the study shows that miscommunication exist in multicultural software teams that is potentially caused by differences in cultures. For example, 68% of the participants indicated that differences in languages (spoken/written) led to miscommunication in the software teams, while 88% of them indicated that differences in individuals' attitudes (behavior, and communication) led to miscommunication in software development teams. Eighty-two percent of the participants agree that differences in values, priorities, or approach lead to miscommunication in software development teams, while 84% of them agree that differences in time management (the way people regard & use time) lead to miscommunication in software development teams. 73% of the participants agree that differences in roles (perceptions of who should make the decisions and who has responsibility for what) lead to miscommunication in software development teams.

Based on the diversity in software teams, 79% of the participants show that miscommunication in requirements engineering phase produces conflicts which increases the amount of rework, while 71% of them agree that miscommunication in the same phase produces conflicts which increases the amount of delays. 62% of the participants agree that miscommunication in multi-cultural teams increases the amount of errors/defects in the final software products. 58% of the participants show that miscommunication in multi-cultural software teams increases the risk of delivering the final product on time. 57% of the participants agree that miscommunication in multi-cultural software teams increases the risk of delivering the final product within budget, while 66% of them agree that miscommunication in multi-cultural software teams affect the overall project productivity.

The second formal study is conducted on software engineers in industry, who are working on a particular software development project in a multi-cultural team or same-cultural team. 34 people participated so far in this study, where 96% of them are working in multi-cultural teams while only 4% are working in same-cultural teams. Around 80% of the participants have more than four years experience working in multi-cultural software teams, and 65% of them have worked on more than 10 projects.

Based on the initial analysis of the data, the study illustrates that people have difficulties expressing ideas, and sometimes fail to interpret messages to teammates in multicultural teams. The study also shows that people in multicultural teams sometimes misunderstand the meaning of the messages conveyed by others, especially in requirements engineering phase. The study also shows that miscommunication exist in multicultural software teams and that caused by differences in cultures. More than 50% of the participants show that miscommunication occurred while working on their projects and the reasons were related to differences in time management, differences in values, priorities, and approach, as well as differences in attitudes, respectively. However, less than 40% of the participants show that miscommunication was related to differences in language and roles, respectively. Based on the diversity, around 76% of the participants show that miscommunication occurred at some level and produced conflicts/errors that led to increasing the amount of rework, while around 80 of them agree that miscommunication occurred at some level and produced conflicts/errors that led to some delays in the project and increased the risk of delivering the final product on time. Around 62% of the participants agree that miscommunication occurred at some level and produced defects/errors in the final software product. Additionally, miscommunication occurred at some level in their projects and produced conflicts/errors that increased the risk of delivering the final product within budget.

Generally, both formal studies show that different cultures have different attitudes, behaviors, values, priorities, and approaches. In addition, those differences have different impacts on the communication process that takes place in requirements engineering, as well as on productivity in terms of rework and delay. In other words, the effect of some cultural factors on communitarian and productivity in software teams vary from culture to another. For some cultures, "time" appears to be a critical factor that could strongly affect communication, while "attitudes" appears to be critical for other cultures. Additionally, different cultures have different miscommunication issues. For example, some cultures have problems in interpreting the messages conveyed by other people, while others have problems expressing those messages.

8 Future Work

This study provides a research body of knowledge from which others can explore the effect of culture on software team management. This research seeks to substantiate the research through a model rather than statistically validate the results due to the cost and timeframe necessary to conduct these studies. However, it provides a research base for future work.

The future work consists of the following steps:

Step 1: Further analyze the data and review the formal studies' results. Consider future sources to refine key areas.

Step 2: Formulate a model based on the initial results and find out how this study could be beneficial for software managers to reason about their multicultural teams. The initial model will provide a map that explains how the differences in cultural factors might affect the communication and the productivity, which helps project managers to be mindful of risks of having miscommunication, which leads to more rework and delay.

9 References

[1] Alkandari, M. "Investigation into Cultural Aspects, Personality, and Roles of Software Project Team Configuration." Computer Science Blacksburg, Virginia Tech Master of Science: 147. 2006.

[2] Boehm, B. "Managing Software Productivity and Reuse" IEEE Computer, Vol. 32, No. 9, pp. 111-113, Sept. 1999.

[3] Brockmann, P., and Thaumuller, T. "Cultural Aspects of Global Requirements Engineering: An Empirical Chinese-German Case Study 2009 4th IEEE International Conference on Global Software Engineering, 2009.

[4] Cunha, A., Canen, A., and Capretz, M. 2009. Personalities, Cultures, and Software Modeling: Questions, Scenarios and Research Directions CHASE'09 - ICSE'09 Workshop, Vancouver, Canada IEEE.

[5] De Souza, R., Sharp, H., Singer, J., Cheng, L., and Venolia, G. 2009. "Cooperative and Human Aspects of Software Engineering" IEEE Software 17-19.

[6] Deresky, H. 2007. International Management: Managing Across Borders and Cultures. United States of America, Addison-Wesley Educational Publishers Inc.

[7] DiStefano, J., and Maznevski, M. 2000. "Creating value with diverse teams in global management" Organizational Dynamics - 2000 Elsevier Science, Inc. 29(1): 45-63.

[8] Gifford, S. 2003. "A Roadmap for a Successful Software Development Team Assembly Model Using Roles." Computer Science and Applications. Blacksburg, Virginia Tech. Master of Science.

[9] Gorla, N., and Lam, Y. 2004. "Who Should Work with Whom? Building Effective Software Project Teams." Communication of the ACM 47(6): 79-82.

[10] Hanisch, J., and Corbitt, B. 2007. "Impediments to Requirements Engineering During Global Software Development " European Journal of Information Systems 16: 793-805.

[11] Hanisch, J., Thanasankit, T., and Corbitt, B. Understanding the Cultural and Social Impacts on Requirements Engineering Processes – Identifying Some Problems Challenging Virtual Team Interaction with Clients Global Cooperation in the New Millennium – The 9th European Conference on Information Systems, Bled, Slovenia, 2001.

[12] John, M., Maurer, F., and Tessem, B. 2005. "Human and social factors of software engineering," ICSE '05 Proceedings of the 27th international conference on Software engineering, New York, NY, USA.

[13] Nash, S., and Redwine, S. 1988. "People and Organizations in Software Production" ACM SIGCPR 11(3): 10-21.

[14] Pressman, R. 2005. Software Engineering - A Practitioner's Approach. New York - USA, Elizabeth A. Jones - McGraw-Hill series in computer science

[15] Stevens, K. 1998. The effects of roles and personality characteristics on software development team effectiveness Computer Science and Applications. Blacksburg, Virginia Tech. Doctor of Philosophy.

AGENT-BASED CMMI FOR SOFTWARE MAINTENANCE PROCESS MEASUREMENT MODEL

Haneen Al-ahmadi, Rodziah Atan, Abdul Azim Abd Ghani and Masrah Azrifah Azmi Murad

Department of Information System, University Putra Malaysia
Serdang, Selangor, 43400, Malaysia
haneeen_0005@hotmail.com & (rodziah, azim, masrah)@fsktm.upm.edu.my

Abstract

The tremendous growth of the area of software measurement requires new model for measurement roles. Changes to software maintenance (SM) process measurement of corrective, preventive, adaptive and perfective maintenance may affect the way of users use the measurement. As well as most of an existing measurement models does not truly or accurately reflect process maturity and tasks difficulty. To verify and solve the current problem issues, we propose a model of SM based on CMMI KPAs (Capability Maturity Model Integration Key Process Areas) and demonstrates these measurement roles into a system of Multi Agent System (MAS) to identify the processes measurement of SM. This model composed of three layers of CMMI layer, Agent layer and SM layer and its architecture consists of four types of agents: Personal Agent (PA), Maintenance Type Agent (MTA), Key Process Area Agent (KPAA) and Ruler Agent (RA). The expected output of our model is to translate the ordinal scale of SM to interval scale to be more accrue and clearly.

Keywords: CMMI model, Multi Agent System, Software Maintenance and Software Maintenance Process.

1 Introduction

Knowledge transfer of a large number of the best practices described in a maturity model has proved difficult [1]. This is especially true during the training of an assessor or a new participant in a process improvement activity.

Software measurement, in order to be effective, must be focused on specific goals; applied to all life-cycle products, process and resources; and interpreted based on characterization and understanding of the organizational context, environment and goals [2].

Software maintenance (SM), according to IEEE definition, is a modification of software product after delivery in order to correct faults, to improve performance or other attributes, to adapt a product to a changed environment, or to improve the product maintainability [3]. Different elements and components of the software maintenance summarized in Figure 1.

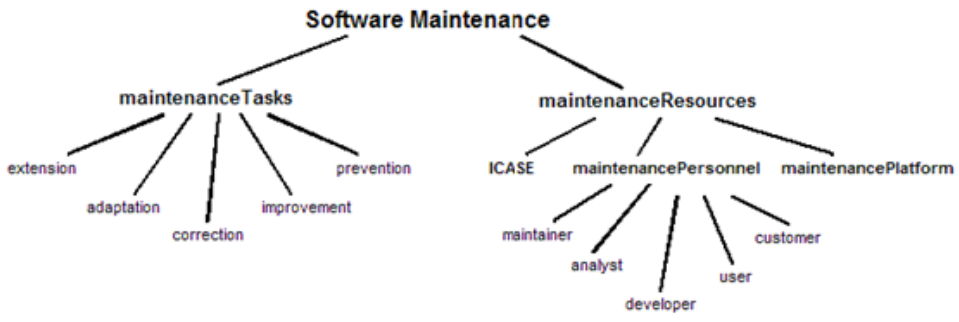


Figure 1. Components of the software maintenance

CMMI-Capability Maturity Model Integration is a tool for implementing best practices for activities concerning product and services in organizations. It was the Software Engineering Institute (SEI) that first began the work with CMMI, and published a book about the subject in 1995. The CMMs have success for many organizations; they have gained increased productivity and quality, and more accurate estimates on time and resource consumption. The CMMs was developed to be applied to different specific areas. Three of these models, the Capability Maturity Model for Software (SWCMM), the Systems Engineering Capability Model and the Integrated Product Development Capability Maturity Model were used as source material to develop the CMMI, the CMMI development team at SEI combined these models and built a framework that can be used in multiple disciplines and that has flexibility to support different approaches [4]. The CMMI is structured in the five maturity levels, the considered process areas, the specific goals (SG) and generic goals (GG), the common features and the specific practices (SP) and generic practices (GP). The *process areas* are defined as follows [5]:

"The Process Area is a group of practices or activities performed collectively to achieve a specific objective."

Such objectives could be the part of requirements management at the level 2, the requirements development at the maturity level 3 or the quantitative project management at the level 4. The difference between the "specific" and the "general" goals, practices or process area is the reasoning in the special aspects or areas which are considered in opposition to the general IT or company-wide analysis or improvement. In this paper the focus is on CMMI for development. The following Figure 2 shows the general relationships between the different components of the CMMI approach.

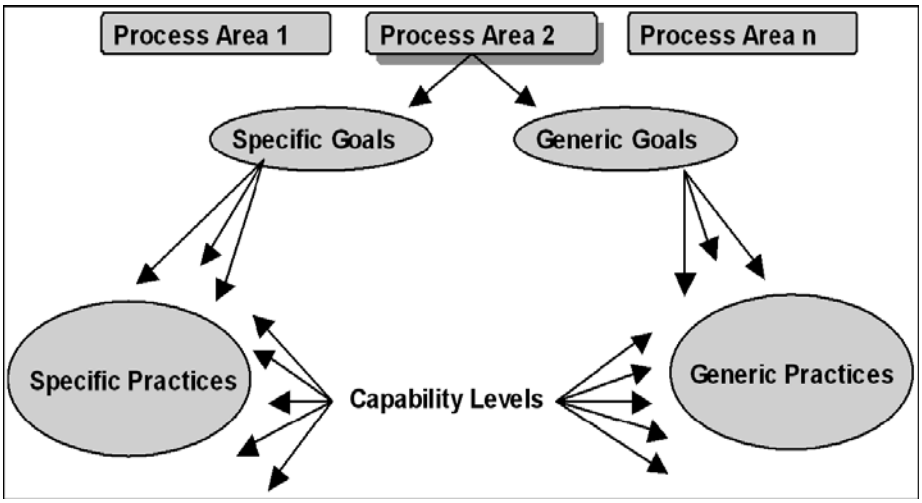


Figure 2. The CMMI model components

To organize the extracted of SM, it is important to classify its information needs. Here, the concept of "*measurement*" is used; this makes it possible to take a snapshot of each maturity level defined in CMMI without going into too much detail [6, 7]. As well as Measurement does not truly or accurately reflect process maturity and tasks difficulty [8]. Changes to SM processes measurement due to corrective, preventive and perfective maintenance may affect the way users use the measurement. This combined CMMI and agent-based tool technique should be properly managed to allow users easier and automated access to measure the SM using an appropriate ruler to measure process maturity.

A multi-agent system (MAS) is a system composed of multiple interacting intelligent agents and it can be used to solve problems which are difficult or impossible for an individual agent or a monolithic system to solve. This has motivated our research to develop a new multi-agent architecture for SM measurement. A SM measurement system is required to monitor SM processes parameters reported by measuring instruments; analyze the measured values; determine if the measurement standards and

procedures are met; and make and execute proper quality measurement action plans in accordance with quality measurement standards, procedures and regulations. This new multi-agent architecture is designed with consideration of the principles of autonomic computing using Prometheus Methodology (PDT).

The main contribution of this paper is; the study shall develop and implement an agent-based CMMI which shall be used to support the measurement for all the types of SM as well as our proposed model shall serve to validate the usage of software agents to assist users to easy measure the SM.

In this paper, in section 2 we present a discussion of the related works. Section 3 provides an overview of our research methodology. Section 4 describes our proposed agent-based CMMI model & its architecture. In, section 5 presents some concluding remarks.

2 Related Works

Many factors must be taken into account before measuring software maintenance processes [9]. One strategy is to identify the key activities of the process. These key activities have characteristics, which can be measured, but, before measures can be identified, it is essential that the software maintenance processes be defined. Software maintenance measurement prerequisites were presented in [10, 11]: 1) definition of maintenance work categories; 2) implementation of a process for requests management; 3) classification of maintenance effort in an activity account data chart (billable/unbillable); 4) implementation of activity management (time sheet) software, data verification; and 5) measurement of the size of change requests.

The SEI [12] describes process measurement activities as appearing at maturity level 2. This confirms the need for a defined process before measurement can be initiated. While the SEI's recommended measures could have been a starting point for maintainers, [3] noted that those measures were created from a development perspective and do not capture the unique features of software maintenance.

Other authors [10, 13] confirm this view, and specify that a software maintenance measurement program must be planned separately from that of the developers: because the measurement requirements are different, software maintenance measures are more focused on problem resolution and on the management of change requests. Higher-maturity organizations have established a maintenance measurement program.

Two different perspectives of maintainability are often presented in the software engineering literature [14]. From an external point of view, maintainability attempts to measure the effort required to diagnose, analyze, and apply a change to specific application software. From an internal product point of view, the idea is to measure the attributes of application software that influence the effort required to modify it. The internal measure of maintainability is not direct, meaning that there is no single measure of the application's maintainability and that it is necessary to measure many sub-characteristics in order to draw conclusions about it [15].

The IEEE 1061 standard [16] also provides examples of measures without prescribing any of them in particular. By adopting the point of view that reliability is an important characteristic of software quality, the IEEE 982.2 Guide proposes a dictionary of measures.

3 Research Methodology

This research shall be carried out in five phases as illustrated in Figure 3

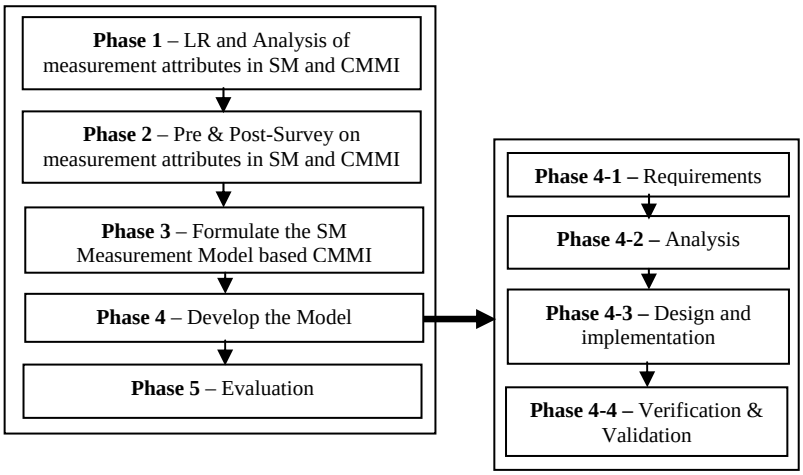


Figure 3. Research Methodology Framework

3.1 Phase 1 – LR and Analysis of measurement attributes in SM, MAS and CMMI

This phase involves evaluation of existing measurement attributes of software maintenance processes and activities and literature reviews on academic journals, software maintenance books, and other tools supporting software maintenance activities.

3.2 Phase 2 – Conduct Pre & Post-Survey on measurement attributes in SM, MAS and CMMI

Questionnaire survey shall be used to evaluate the different maintenance measurement models used by selected SM organizations in some universities, Kingdom of Saudi Arabia. The survey shall cover collaborative activities and knowledge required within the SMP and CMMI measurement tools. The stage of post-survey will be collected and analyzed after the development of the system.

3.3 Phase 3 – Formulate the SM Measurement Model based CMMI

The next step is to formulate new measurement model of applying MAS based CMMI for collaborative software maintenance based on earlier literature reviews and pre-survey results

3.4 Phase 4 – Develop the Model

The model development followed the software development life cycle (SDLC) that include the following steps: Requirements, Analysis, Design and implementation and Verification & Validation.

3.4.1 Phase 4-1 – Requirements

Our proposed system will be implemented using Java Programming Language and NetBean IDE 6.1.

3.4.2. Phase 4-2 – Analysis

The Measurement and Analysis process area supports all process areas by providing specific practices that guide projects and organizations of software maintenance in aligning measurement needs and objectives with a measurement approach that will provide objective results.

These results can be used in making informed decisions and taking appropriate corrective actions. The purpose of Measurement and Analysis is to develop and sustain a measurement capability that is used to support management information needs.

3.4.3 Phase 4-3 – Design and implementation

Based on the identified SM processes and activities, the MAS design shall be specified to determine the types of agents, events, protocols and agent capabilities, using the Prometheus methodology [17]. The methodology has a good modeling tool and can be used to generate initial codes that can be used by Agent-oriented tools such as Jack Agent development tool. Prometheus steps are as follows:

1. *Systems specification* – identifies goals, use case scenarios, functionalities, actions (outputs) and percepts (inputs).

2. *Architectural design* – determine agent types based on data coupling, agent acquaintance and interaction diagram.

3. *Detailed design* – refine the agent descriptions by defining and describing the capabilities and plans.

The next step is to develop the prototype of Agent-Based CMMI. The platform and development tool shall be determined at a later stage.

3.4.4 Phase 4-4 – Verification & Validation

This is an iterative process, whereby the tools are tested and reworks and enhancements are carried out. We expect several builds are required to stabilize the application.

3.5 Phase 5 – Evaluation

Data evaluation is proposed for each of the following dimension:

- ✓ Different SM types
- ✓ Different SM process and applications

4. Proposed Agent-Based CMMI Model & Its Architecture

4.1 Proposed Agent-Based CMMI Model

This section describes our system model framework as illustrated in figure 4. Figure 4 below shows a schematic representation of our system model. The framework has been built by using three layers.

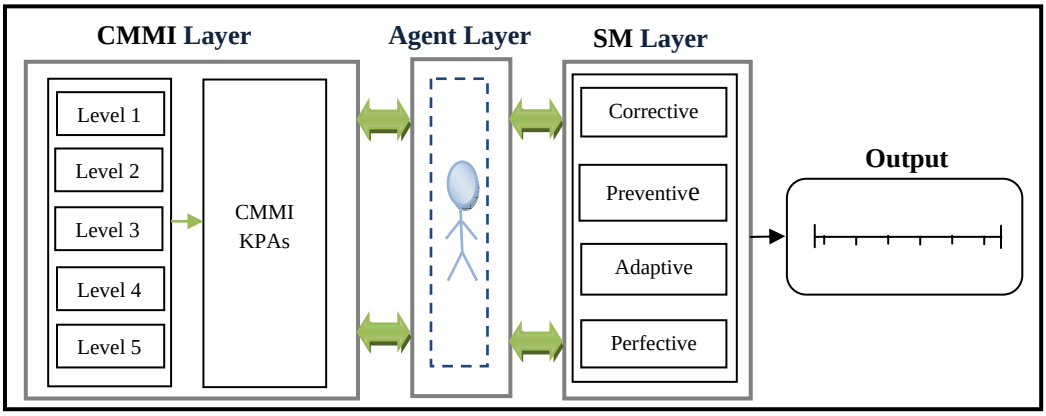


Figure 4. Proposed Agent-Based CMMI Model

The functionality of those layers can be summarized as:

- **CMMI Layer:** this layer is consists of level maturity model of CMMI as well as the key process areas (KPA) of each level to provide SM by the measurement rules. This layer has one agent: the Key Process Area Agent (KPAA). KPPA provide the system by the KPA of CMMI to measure the software maintenance type.

- **Agent Layer:** This layer has one agent: the User Personal Agent (PA). PA acts as an effective bridge between the user and the rest of the agents. Such agents actively assist a user in operating an interactive interface.

- **SM Layer:** This layer has two agents: Maintenance Type Agent (MTA) and Ruler Agent (RA). MTA provide the system by the type of software maintenance that willing to be measured. RA used to translate the output of the measurement scale from the number to the ruler measurement style.

4.2 Proposed MAS Architecture

The proposed agent-based CMMI model architecture composed of four types of agents described as follows:

4.2.1 Personal Agent (PA): Main responsibility of this agent is acts as an effective bridge between the user and the rest of the agents. Such agents actively assist a user in operating an interactive interface.

4.2.2 Maintenance Type Agent (MTA): Main responsibility of this agent is to provide the system by the type of software maintenance that willing to be measured.

4.2.3 Key Process Area Agent (KPAA): Main responsibility of this agent is to provide the system by the KPA of CMMI to measure the software maintenance type.

4.2.4 Ruler Agent (RA): Main responsibility of this agent is to translate the output of the measurement scale from the number to the ruler measurement style.

Conclusion

Measuring the concepts most often selected by software maintenance users is helpful in identifying where our effort should be spent in detailing both the support tools and the maturity model. The next step in this research project is to implement our proposed model, validate the results with experts in the domain and determine whether or not the measurement of our proposed model usage is useful for identifying the most important maintenance preoccupations of today's maintainers. CMMI provides examples of different measurement objectives like reduce time to delivery, reduce total lifecycle cost, deliver specified functionality completely, improve prior levels of quality, improve prior customer satisfaction ratings, etc. Achievement of all these objectives can be measured through continuous monitoring of the SMP performance. Therefore, the implementation of CMMI Measurement based on MAS architecture described in this paper is based on the assumption that the main measurement objective is to support the measurement for all the types of SM as well as to validate the usage of software agents to assist users to easy measure the SM. Our proposed model composed of three layers of CMMI layer, Agent layer and SM layer and its architecture consists of four types of agents: Personal Agent (PA), Maintenance Type Agent (MTA), Key Process Area Agent (KPAA) and Ruler Agent (RA).

References

[1] A. Abran. J. Moore. W. Bourque. P. Dupuis., and L. Tripp.: *Guide for the Software Engineering Body of Knowledge (SWEBOK)*, Ironman version, IEEE Computer Society Press: Los Alamitos CA, 6-1-6-15, 2004

[2] V.R. Basili, G. Caldiera, and H.D. Rombach.: *The Goal Question Metric Approach.* In Encyclopedia of Software Engineering, Wiley, pp. 528-532, 1994

[3] T.M. Pigoski.: Practical Software Maintenance: Best Practice for Managing your Software Investment, Wiley, 1997

[4] M.B, Chrissis. M. Konrad, and S. Shrum.: *CMMI second edition Guidelines for process integration and product improvement.* Pearson Education, Inc, 2007

[5] M.K. Kulpa., and K.A. Johnson.: *Interpreting the CMMI – A Process Improvement Approach.* CRC Press Company, 2003

[6] CMMI PRODUCT TEAM, *Capability Maturity Model Integration for Development (CMMI-DEV, V1.2)*, CMU/SEI-2006-TR-008, Technical Report, Software Engineering Institute, Pittsburgh (PA), August 2006.

[7] A. Abran. AL Qutaish, R., Desharnais, and N. Habra.: *An Information Model for Software Quality Measurement with ISO Standards*, International Conference on Software Development – SWDC-REK, University of Iceland, Reykjavik, Iceland, May 27-June 1, 2005.

[8] W.S. Humphrey.: *Managing the Software Process*, Addison-Wesley Publishing Company, Reading, Massachusetts, pp. 494, 2009

[9] S.L. Pfleeger., and S.A. Bohner: A framework for maintenance metrics. Proceedings of the Conference on Software Maintenance (Orlando, FL), IEEE Computer Society Press, 1990

[10] J.M. Desharnais, F. Par, and D. St-Pierre.: Implementing a Measurement Program in Software Maintenance – an Experience Report Based on Basili's Approach, IFPUG Conference, Cincinnati, OH, 1997

[11] A. April., and D. Al-Shurougi.: Software Product Measurement for supplier Evaluation, Proceedings of the Software Measurement Conference (FESMA-AEMES), Madrid, Spain, October 18-20, 2000, Http://www.gelog.etsmtl.ca/publications/pdf/583.pdf [February 10, 2008].

[12] Capability Maturity Model Integration for Software Engineering (CMMi). Version 1.1, CMU/SEI-2002-TR-028, ESC-TR-2002-028, Software Engineering Institute, Carnegie Mellon University, 2002

[13] J. McGarry.: Practical Software Measurement: A Guide to Objective Program Insight, Department of Defense, September 1995.
[14] B. Lagu., and A. April.: Mapping of the ISO9126 Maintainability Internal Metrics to an industrial research tool, Proceedings of SES 1996, Montreal, October 21-25, http://www.lrgl.uqam.ca/sponsored/ses96/paper/lague.html, 1996
[15] R.S. Pressman.: Software Engineering: A Practitioner's Approach. McGraw-Hill, New York, NY, pp. 860, 2001
[16] Institute of Electrical and Electronics Engineers. IEEE Standard for a Software Quality Metrics Methodology, Standard 1061-1998. Institute of Electrical and Electronics Engineers: New York, NY, pp. 35, 1998
[17] L. Padgham., and M. Winikoff.: Developing Intelligent Agent Systems. John Wiley and Sons, 2004

Code Rocket: Seeking improvements in detailed design support for non-model driven approaches to development

C. Ramsay[1], **S. Parkes**[1], **and A. Spark**[2]
[1]School of Computing, University of Dundee, Dundee, Scotland, UK
[2]Rapid Quality Systems Limited, Dundee, Scotland, UK

Abstract - *Code Rocket is a support tool for software developers which allows them to perform the detailed design of a software system and produce program code simultaneously. It integrates with existing development environments to allow developers to visualize program code during code construction and code maintenance. Code Rocket supports other project stakeholders, including non-programmers, by providing improved visibility of the detailed stages of code development in a way that is quick and easy to assimilate. Code Rocket seeks to bridge the detailed design gap between UML and code in mainstream software development, for projects which do not adopt formal or model-driven approaches. This paper introduces Code Rocket, describes the background and rationale for its development and presents the results of a survey of industry users which suggest that it may offer benefits for improving visibility of business logic within code, assisting comprehension of code, and helping developers to become productive more quickly*.

Keywords: Detailed design, documentation, tool support, code maintenance, program comprehension.

1 Introduction

A wide range of tools are available to assist with the design and documentation of software systems. These include Unified Modeling Language (UML) [1] tools which provide support for modeling the higher level structure and behavior of software systems using Class diagrams, Sequence diagrams, and various other UML diagram types. However, in mainstream software development, there can be a subsequent gap in design support between the stages of completing the higher level design and then implementing the detailed algorithms and processes required in the actual code. Effective support for the detailed design of algorithms may be missing or requires use of diverse tools which are not optimized for use as part of a streamlined development process. Such tools may often require design information to be held separately from the code that it describes, thereby incurring additional overheads for ensuring designs and code remain synchronized with respect to changes in each other.

2 Documentation tool support

Many tools exist to automate the process of generating documentation directly from program code and these fit well into automated build processes. For example tools such as Doxygen and Javadoc. These tools typically only offer retrospective views of the code after it has been written rather than providing up-front design support during code construction for complex algorithms and processes. Capabilities for up-front and roundtrip design and documentation are facilitated in other tools but tend to focus on higher level structures such as class diagrams, in some cases sequence diagrams. For example UML tools such as IBM Rational Software Modeler, Sparx Enterprise Architect, and increasingly within development environments such as Microsoft Visual Studio and Eclipse which incorporate various UML plug-ins and features.

Tools which provide method-level design documentation such as Doxygen tend to limit themselves to extracting documentation from 'header' comments above structures in code. Header comments provide a valuable summary of the purpose of methods in code, including design trade-offs and considerations. However they do not offer visualization of the actual steps which have been implemented inside the body of the method, which could include complex algorithms, important business processes and safety critical processes which have to be conducted in very precise, prescriptive sequences. It is proposed that visualization of such internal processes offers improved support for software developers when designing and reviewing them. It also supports review and discussion amongst the wider stakeholders on a software project, including non-programmers whose requirements the software development effort is seeking to address. However, such levels of detailed documentation could incur yet further layers of overheads to create and maintain which may hinder their cost effectiveness and adoption as part of a streamlined development process.

Other tools exist to reverse engineer design views of the internal steps of methods, allowing visualization in forms such as flowcharts. For example tools such as Visustin and Code Visual to Flowchart. Many of these are only loosely integrated into existing tools and processes, e.g. in some cases requiring transferring program code from one environment to another to

achieve design visualization. However batch processing of code files can usually be achieved too. Detailed design support is also available in the form of plug-ins to development environments. For example, Microsoft's Visual Studio Learning Pack 2.0 offers generation of static flowcharts from code. Typically such tools focus on provision of retrospective documentation and don't provide up-front design or roundtrip capabilities for fully automated/dynamic detection and synchronization of changes in code and designs.

3 Formal approaches

Formal or model-driven approaches to development [2] inherently address the detailed design gap between UML and code through use of intensive modeling of both the structure and behavior of software systems, including the possibility of executable models. Studies of the use of domain specific languages have indicated significant benefits in areas such as productivity, quality and ease of maintenance [3, 4, 5]. Model-driven approaches are typically adopted in specific industry sectors such as telecommunications and embedded systems development and haven't yet gained wide acceptance in practice. They still face challenges in terms of adoption costs, the maturity of tooling and also a people and process perspective [6, 7, 8]. The design and modeling notations adopted in formal and model-driven approaches also typically require specific expertise which may limit their comprehension amongst wider members of a project team for communication and review. Model-driven approaches strive to make software development a forward only process, e.g. code is automatically produced from models and future changes to code should be applied to models first and then forward engineered into code, and not the other way around. In reality changes are frequently made directly to code itself which renders the original models that may have produced that code out of date. This may often be a result of model-driven approaches that haven't been applied fully and/or the extent of adoption underestimated which results in a hybrid of partial model-driven processes with traditional programming practices.

Code Rocket is a software tool which seeks to provide an alternative to model-driven processes for detailed design support where traditional programming is still prevalent. It seeks to support the way that software developers work in practice by allowing them to make changes to code without fear of associated designs becoming out of date whilst retaining the value of up-front design support. Code Rocket also seeks to provide support for the maintenance of the large amount of legacy code in existence which pre-dates the emergence of model-driven approaches.

4 Code Rocket

Code Rocket is a software design and code visualization system developed within the Space Technology Centre in the School of Computing which aims to bridge the detailed design gap between UML and code by providing automated, detailed design documentation support for software developers. Code Rocket's design support is embedded directly within popular development environments used by software developers to seamlessly integrate into their current working processes without hindering productivity and to eliminate design documentation overheads. Code Rocket allows detailed designs to be produced as a natural by-product of the development process. Design information can be extracted entirely from code, therefore preventing the need to maintain design documentation separately from code. Code Rocket not only extracts documentation at the header level of code constructs but delves deeper into the internal algorithms of the methods in code, allowing them to be visualized in both textual and graphical forms which are accessible to both programmers and non-programmers on a software project; therefore supporting wider review, discussion and quality assurance. Code Rocket fully automates the process of generating detailed designs and documentation from code and employs various labor saving strategies to streamline the translation of up-front designs into code; including code forward engineering and automated generation and layout of flowchart diagrams. This eliminates the often fiddly and time consuming process required to create and edit diagrammatical representations. Code Rocket's design views are fully synchronized with respect to each other and the code they describe, allowing developers to switch between code and design views at any time and ensure that they retain an up-to-date reflection of their content and streamlining cognitive processes during code construction and comprehension tasks.

Figure 1 provides an example of the Code Rocket plug-in to Microsoft Visual Studio to design and/or visualize a payment process which is being implemented as part of an online banking application. Code Rocket provides two complementary design views in the form of pseudocode and flowcharts to assist with the upfront design of detailed, algorithmic software logic. These can be used according to the user's preference. For example, pseudocode may be used by programmers because it is not too far distanced from the code whilst the flowchart may be more suited to stakeholders that are not programmers for a more abstract view. However these views can be used interchangeably and are automatically synchronized with each other. The flowchart and pseudocode design views are available within a standalone application (to support upfront design or legacy code inspection) and also embedded as plug-ins to development environments such as Visual Studio and Eclipse (to provide automated design and code visualization support upon demand). Figure 1 shows the Code Rocket design views embedded inside Visual Studio.

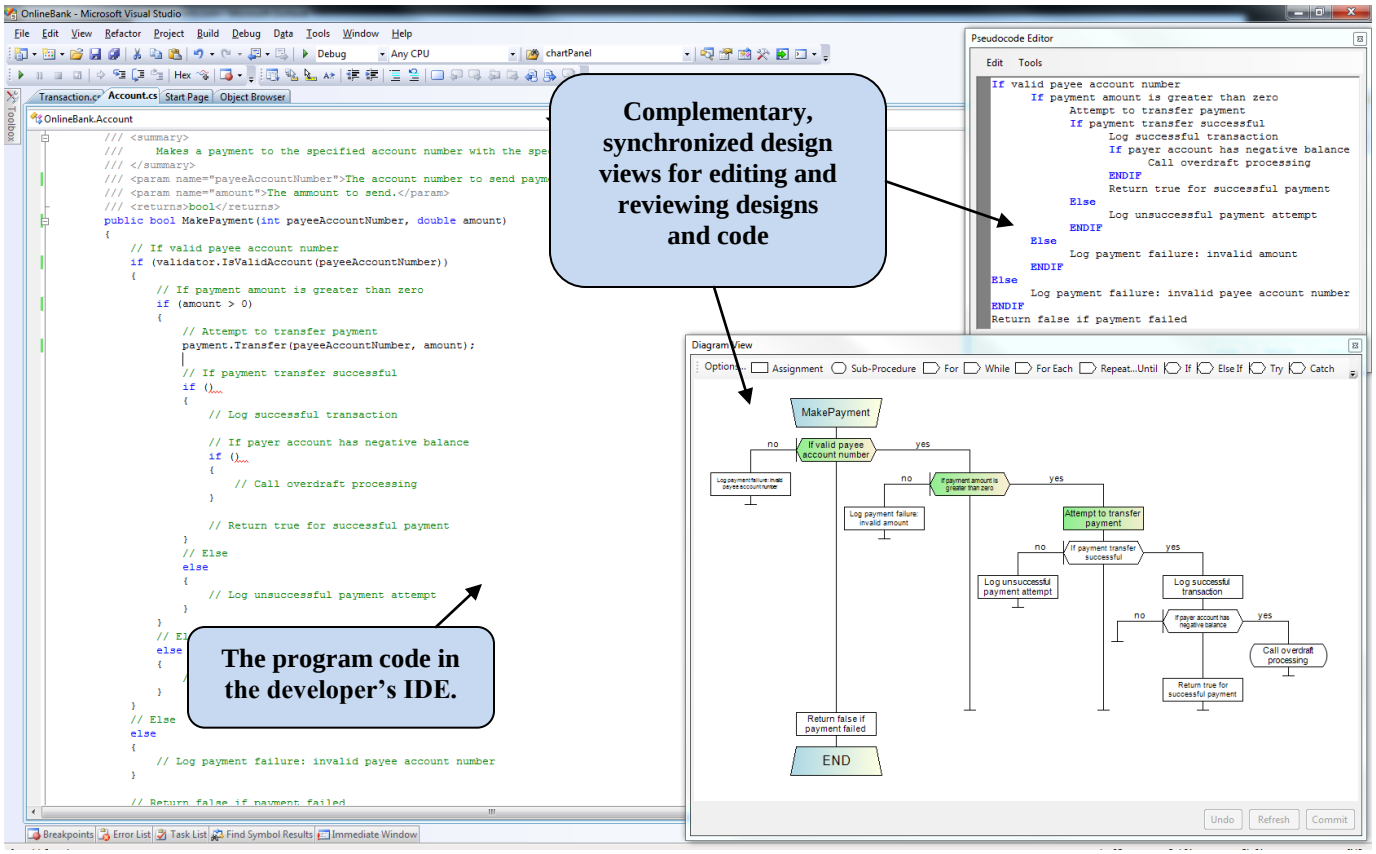


Figure 1. Screenshot of the Code Rocket plug-in for Microsoft Visual Studio.

The choices of detailed design views provided in Code Rocket were driven by the desire to serve the needs of both programmers and non-programmers on a software project by providing them with easily recognizable forms of documentation both in plain language (pseudocode) and business process-type flowcharts. This would allow software developers to design and visualize detailed control flow logic in way that is easily understandable to others, to assist communication and review and expose visibility of business logic in code. Diagrams, such as flowcharts, have been shown to support faster searching of information and subsequent inferences made during cognitive processing by exploiting the layout, relationships, and/or groupings of visual elements [9]. They are able to aid comprehension by reinforcing the key ideas and restricting the possibility of varied interpretations, if they are well matched to the structure of the problem they represent [10, 11, 12]. Shneiderman et al [13] found no significant benefits to the use of flowcharts over any of the other forms of documentation for a series of code comprehension activities. However there may have been flaws in the experiment design [14, 15, 16]. Scanlan [16] found that flowcharts, compared to pseudocode, took less time to comprehend, produced fewer errors in understanding, gave subjects higher confidence in the accuracy of their answers, and reduced the time required to provide answers. Crews and Zeigler [17] achieved similar results but Whitley [18] highlights several issues which may have biased the results achieved by Scanlan. Ramsey et al [19] found significantly better quality and more detailed designs when using a program design language, a form of pseudocode, compared to flowcharts. Curtis et al [14] concluded that a form of constrained sequential language/pseudocode typically outperformed other forms of documentation, closely followed by a form of constrained branching language and branching ideograms (essentially a flowchart-type representation). Green and Petre [20] found no significant advantage of visual programs over textual programs. In fact, in many cases, the visual programs were assessed as being harder to read. Chattratichart and Kuljis [21] on the other hand, demonstrated the superiority of visual representations in both control-flow and data-flow paradigms in terms of response time and accuracy for a series of tasks investigating novice programmers. In non-programming experiments representations such as flowcharts have demonstrated superiority in comprehension and accuracy compared to printed instructions [22] and for producing fewer errors in problem solving tasks, but with a possible deterioration effect depending upon conditions of use [23].

Although there have been difficulties in formulating conclusive empirical evidence to support a unanimous view; there continues to be widespread anecdotal belief for the benefits that both diagrammatic and textual programming and design representations provide for tasks such as reasoning,

problem solving, and computation. As a result, Code Rocket provides support for both textual (pseudocode) and diagrammatical (flowchart) forms of design documentation in order to exploit their combined strengths together. In order to counter the additional overhead of maintaining two separate forms of documentation, various labor saving devices are employed. For example flowcharts are automatically generated from the corresponding pseudocode designs or directly from code. This saves effort for what are usually time consuming and fiddly graphical editing tasks. Skeleton code can be forward engineered from designs. Changes in design views are automatically synchronized so that the user can switch between them to work within different cognitive contexts without worrying about designs getting out of date. Changes in designs are also merged into code, so that code and designs remain synchronized. Various strategies are offered to manage larger diagrams, such as zooming, collapsing and expanding sections of flowcharts, and grouping sections of flowcharts into higher level elements which can subsequently be expanded into when required.

Code Rocket makes use of comments within code to enhance readability of the abstract design views that it generates. Therefore, well-commented code will naturally produce highly readable pseudocode and flowcharts. However, when user-written comments are not available, the code itself is analyzed to produce English-readable equivalents to include within the design representations. For example, the code extract in Figure 2 below, would result in the flowchart design shown in Figure 3 being automatically produced.

```
// if valid payee account
if (AccountsServices.IsValidAccount(payeeAccount))
{
    if (paymentAmount > 0)
    {
        // attempt to transfer payment into account
        PaymentTransferResult result =
                    Transfer(amount, payeeAccount);
    }
}
```

Figure 2. Code sample.

The intention of the design views is to provide an alternative, abstract visualization that is an entry point into the code that the developers have to comprehend and that project managers may wish to review. This allows them to quickly build an initial mental model of what the code is doing, including the various conditions and pathways it contains but without being distracted by the programming language syntax and constructs. When the developer subsequently moves into the code, they have already formed an abstract model of its content and purpose. This then acts as a form of 'sign posting' to guide them through the actual code itself.

The generation of English-readable equivalents of uncommented code for presentation in design views cannot always guarantee a perfect outcome. If they are not sufficient to aid understanding, the actual program code itself can be overlaid in Code Rocket's flowchart view instead. This can assist developers who may wish to view the detail in the code but also benefit from viewing the flow of control and branching conditions graphically.

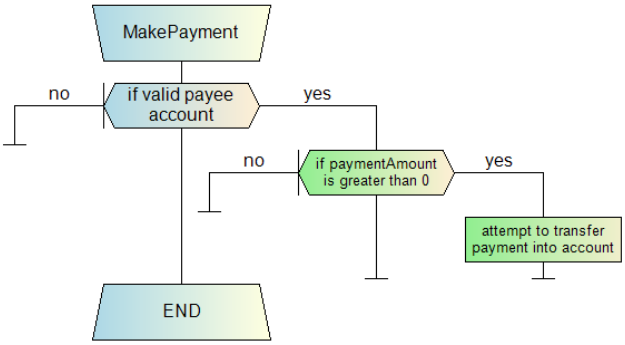


Figure 3. Resultant design for sample code.

5 Benefits and results

Code Rocket is a focus of ongoing research and development within the Space Technology Centre at the University of Dundee. The research seeks to determine the effectiveness of providing automated, integrated, fully synchronized detailed design documentation support to deliver cost and quality benefits during application development. In order to measure the possible benefits that Code Rocket may deliver a survey has been conducted using industrial partners evaluating the Code Rocket technology. These partners encompass a range of industrial sectors including computer games, the space industry, computer simulation/modeling, and psychometric test and evaluation systems. The industrial partners were provided with the Code Rocket software and asked to adopt it for tasks they were undertaking as part of their current, ongoing development projects for a period of 1 month or more. A survey questionnaire was constructed to ask participants for their views regarding the use of Code Rocket in various areas of software development such as design, documentation, code maintenance, debugging, project management, review, and customer involvement. The respondents were asked to rate the extent to which they believed that Code Rocket could or did offer benefits and/or cost and quality savings in these various areas. Approximately 15 questionnaires were issued of which 9 responses were received representing a cross section of various software development roles such as programmers, senior programmers, system designers/architects, and CTO-level roles. A summary of the results are presented in Table 1 below.

Table 1 presents the average savings in time or cost that survey participants indicated they would expect to achieve by using Code Rocket for various software development tasks. The results indicate the potential for significant savings in areas such as detailed design documentation, code

maintenance tasks, and communication amongst diverse project stakeholders.

Table 1. Results of initial Code Rocket survey.

Task	Average saving in time or effort expected from using Code Rocket
Detailed software design	6-10%
Code construction	1-5%
System documentation	11-20%
Debugging	1-5%
Project management	1-5%
Communication between diverse stakeholders	6-10%
Code comprehension and maintenance	11-20%

Survey participants also made the following more general observations of the actual and/or perceived benefits that may be delivered by Code Rocket:

- Improved visibility of business logic in code.
- Better communication across development teams of system logic to help reduce misunderstandings and gain agreement more efficiently.
- Eased sharing of code knowledge throughout a team.
- Enhanced quality of commenting of code.
- Improved support for code comprehension, helping new developers (or developers moving to unfamiliar areas of code) to become productive more quickly.

Further experiments and studies are planned to continue to ascertain these benefits and also in other areas of the software development process such as team communication, requirements gathering, and project management. The results of any such studies will be presented at SERP'11 if available.

As a result of successful initial trials the Code Rocket technology has subsequently been commercialized by a spin-out company from the Space Technology Centre (Rapid Quality Systems Ltd [24]), and is currently being used by businesses in a variety of sectors.

6 Conclusion

In mainstream software development there is a gap in detailed design tool support which exists between the stages of performing the higher level design of a software system and implementing the detailed algorithms in code. Existing tools either: (i) focus on retrospective documentation after coding rather than providing up-front support; (ii) aren't seamlessly integrated into development processes, therefore incurring overheads and hindering productivity; (iii) aren't in mainstream adoption, such as model-driven approaches to detailed design. Developers may have to resort to jumping directly into code without adequate opportunity to resolve detailed design issues and with no opportunity to obtain review and discussion of detailed business logic across the wider project team, including non-programmers.

Code Rocket is a code visualization and documentation system which aims to bridge this gap in detailed design support between UML and code through provision of automated, integrated, intuitive tool support in the form of pseudocode and flowcharts which are automatically synchronized with corresponding code in software development IDEs such as Microsoft Visual Studio and Eclipse. The results of a survey of industrial partners evaluating Code Rocket indicate benefits in areas such as detailed design, documentation, and assisting with code comprehension and maintenance activities.

The authors would like to acknowledge the support of Scottish Enterprise under the Proof of Concept and SMART programmes.

7 References

[1] Fowler M. and Scott K. 2003. UML Distilled: A Brief Guide to the Standard Object Modeling Language. Addison Wesley.

[2] Stahl T. and Voelter M. 2006. Model-Driven Software Development: Technology, Engineering, Management. Wiley.

[3] Kärnä J., Tolvanen J.-P., and Kelly S. 2009. Evaluating the Use of Domain-Specific Modeling in Practice, In Proceedings of the 9th OOPSLA workshop on Domain-Specific Modeling

[4] Hammond, J. 2008. Boosting productivity with domain-specific modeling. Micro Technology Europe, January 2008, pp 10-15

[5] Cao L., Ramesh B., and Rossi M. 2009. Are Domain-Specific Languages Easier to Maintain Than UML Models? IEEE Software, vol. 26, no. 4, 19-21.

[6] Mohagheghi P. and Dehlen V. 2008. Where is the proof? – A review of experiences from applying MDE in

industry. In: Proceedings of the 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08), pp. 432–443.

[7] Teppola S., Parviainen P., and Takalo J. 2009. Challenges in the Deployment of Model Driven Development. Fourth International Conference on Software Engineering Advances (ICSEA '09), pp 15-20.

[8] Teppola S. and Parviainen P. 2011. Model Driven Development – Attention to People and Processes. Proceedings of the IASTED International Conference on Software Engineering (SE 2011), pp 183-190.

[9] Larkin, J.H., Simon, H.A. 1987. Why a Diagram is (Sometimes) Worth Ten Thousand Words. Cognitive Science, vol. 11, 65-99.

[10] Gilmore, D.J., Green, T.R.G. 1984. Comprehension and Recall of Miniature Programs. International Journal of Man-Machine Studies, vol. 21, 31-48.

[11] Gurr, C.A. 1999. Effective Diagrammatic Communication: Syntactic, Semantic and Pragmatic Issues. Journal of Visual Languages and Computing, vol. 10, no. 4, 317-342.

[12] Stenning, K., Oberlander, J. 1995. A cognitive theory of graphical and linguistic reasoning: Logic and implementation. Cognitive Science, vol. 19, no. 1, 97-140.

[13] Shneiderman, B., Mayer, R., McKay, D., Heller, P. 1977. Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM, vol. 20, no. 6, 373-381.

[14] Curtis, B., Sheppard, S.B., Kruesi-Bailey, E., Bailey, J., Boehm-Davis, D.A. 1989. Experimental evaluation of software documentation formats. Journal of Systems and Software, vol. 9, no. 2, 167-207.

[15] Fenton, N., Pfleeger, S.L., Glass, R.L. 1994. Science and Substance: A challenge to software engineers. IEEE Software, vol. 11, no. 7, 86-95.

[16] Scanlan, D.A. 1989. Structured flowcharts outperform pseudocode: An experimental comparison. IEEE Software, vol. 6, no. 5, 28-36.

[17] Crews, T., Zeigler, U. 1998. The Flowchart Interpreter for Introductory Programming Courses", In: Proc. of FIE '98 Conf., Tempe, Arizona, USA, 307-312

[18] Whitley, K.N. 1997. Visual Programming Languages and the Empirical Evidence For and Against. Journal of Visual Languages and Computing, vol. 8, no. 1, Feb., 109-142.

[19] Ramsey, H.R., Atwood, M.E., Van Doren, J.R. 1983. Flowcharts vs. Program Design Languages: An Experimental Comparison. Communications of the ACM, vol. 26, no. 6, 445-449.

[20] Green, T.R.G., Petre, M. 1992. When Visual Programs are Harder to Read than Textual Programs, In: Proc. Sixth European Conf. Cognitive Ergonomics. (ECCE-6), 167-180.

[21] Chattratichart, J., Kuljis, J. 2002. Exploring the effect of control flow and traversal direction on VPL usability for novices. Journal of Visual Languages and Computing, vol. 13, no. 5, 471-500.

[22] Kamman R. 1975. The comprehensibility of printed instructions and the flowchart alternative. Human Factors, vol. 17, pp 183-191.

[23] Wright P. and Reid F. 1973. Written information: Some Alternatives to Prose for Expressing the Outcomes of Complex Contingencies. Journal of Applied Psychology, vol. 57, no. 2, pp. 160-166.

[24] Rapid Quality Systems. www.rapidqualitysystems.com

Model Transformation for a System of Systems Dependability Safety Case

Judy Murphy

MPL Inc.
832 E. Woodland Tr.
Prairie du Sac, WI 53578
USA

jmurphy@mpl.com

Stephen B. Driskell

TASC Inc.
1000 Technology Dr.
Fairmont, WV 26554
USA

Stephen.Driskell@TASC.COM

Abstract *– Software plays an increasingly larger role in all aspects of NASA's science missions. This has been extended to the identification, management and control of faults which affect safety-critical functions and by default, the overall success of the mission. Traditionally, the analysis of fault identification, management and control are hardware based. Due to the increasing complexity of system, there has been a corresponding increase in the complexity in fault management software. The NASA Independent Validation & Verification (IV&V) program is creating processes and procedures to identify and incorporate safety-critical software requirements along with corresponding software faults so that potential hazards may be mitigated.*

This Specific to Generic … A Case for Reuse paper describes the phases of a dependability and safety study which identifies a new process to create a foundation for reusable assets. These assets support the identification and management of specific software faults and, their transformation from specific to generic software faults. This approach also has applications to other systems outside of the NASA environment.

This paper addresses how a mission specific dependability and safety case is being transformed to a generic dependability and safety case which can be reused for any type of space mission with an emphasis on software fault conditions.

Keywords: Reuse, Safety, Dependability, Validation, Verification, Model, Transformation

1 Introduction

The National Aeronautics and Space Administration (NASA) have a portfolio of major projects. These range from highly complex and sophisticated space transportation vehicles, to robotic probes, to earth orbiting satellites equipped with advanced sensors. In many cases, NASA's projects are expected to incorporate new and sophisticated technologies that must operate in harsh, distant environments. These projects have also produced ground-breaking research and advanced our understanding of the universe. However, one common theme binds most of the projects - they cost more and take longer to develop than planned [1].

Many of these systems function together as complex software intensive, safety-critical systems of systems (SoS) to support NASA's research missions in science, aeronautics, and human space flight exploration [2]. A SoS consists of multiple components and subsystems. A SoS development project is usually accomplished over a period of several years and most likely has rules, regulations and standards that must be followed. In NASA's case, it's imperative that all projects, including the SoS adhere to NASA's Office of Safety and Mission Assurance (OSMA) policies and procedures.

The mission of NASA's IV&V program, under the auspices of the OSMA, is to provide the highest achievable levels of mission and safety-critical software [9]. Safety assurance ensures that the requirements, design, implementation, verification and operating procedures for the identified software minimizes or eliminates the potential for hazardous conditions [3]. Software safety activities occur within the context of system safety, system development, and software development and assurance [3]. System safety assessment is a disciplined, systematic approach to the analysis of risks resulting from hazards that can affect humans, the environment, and mission assets [4]. The NASA IV&V Program provides assurance to our stakeholders and customers that NASA's mission-critical software will operate dependably and safely [2].

The IV&V program identified a need to address software-centric safety analysis and assess the quality of software safety engineering early in the development of a SoS to ensure the software manages safety requirements while not introducing system hazards. Most of the analysis is conducted via manual inspection, source code analysis, and more recently independent testing. The complex nature of software intensive systems introduces the challenge of narrowing the gap between the system requirements and their implementation. In spite of efforts to improve the flow of engineering information throughout the development process, oftentimes the implemented system does not fully match the required one, nor does it meet the user needs and

expectations [5]. This discontinuity between requirements documentation, software, and design implementation prevents sensible reusability, especially when the analysis is hardware specific.

The IV&V team develops its own independent understanding of each system under development which consists of a custom System Reference Model (SRM). These SRMs consist of a set of use cases, activity and sequence diagrams using the Unified Modeling Language (UML).

The basis of the modeling activity is derived from the review of artifacts provided by the SoS developer for each project. These artifacts include, but are not limited to, operations concepts, requirements, specifications, code, Failure Modes and Effects Analysis (FMEA), Fault Management (FM) and Failure Fault Analysis (FFA).

The IV&V analyses are model-based, striving to obtain goodness of product data in terms of three questions: *What is the system software supposed to do? What the system software is not supposed to do? What is the system software's expected response under adverse conditions?*

During Phase I of a multi-phase dependability and safety study, the SRM was extended to provide additional verification and validation. Specifically, for the spacecraft safe-hold (which is the autonomous software for managing spacecraft hazards without ground intervention) which establishes "safe" stable spacecraft operation, with minimal power consumption, power-positive (sun pointing), battery charging, and (for an earth orbiting mission) omni-directional (ground) communication. Ground operations can recover from the safe event by assessing the spacecraft failure and providing intervention to restore mission operations [2]. Safe-hold satisfies the cardinal rules for safety-critical software which are [6]:

- No single event or action shall be allowed to initiate a potentially hazardous event.
- When an unsafe condition or command is detected, the system shall:
    - Inhibit the potentially hazardous event sequence.
    - Initiate procedures or functions to bring the system to a predetermined "safe" state.

For the purpose of this paper, dependability and software safety are defined as:

Dependability:

- Dependability is the degree to which an item is capable of performing its required function at any randomly chosen time during its specified mission operating period, disregarding scheduled maintenance outages.

Software Safety:

- Software Safety is the aspect of software engineering and software assurance that provide a systematic approach to identifying, analyzing, and tracking software mitigation and control of hazards and hazardous functions (e.g., data and commands) to ensure safer software operation within a system [3].

The focus of the Phase I effort was a science satellite mission, with mature artifacts and a fairly comprehensive list of hardware fault conditions.

This paper addresses how a mission specific dependability and safety case is transformed to a generic dependability and safety case which can be reused for any type of space mission with an emphasis on software fault conditions.

The organization of the paper is as follows. Section 2 presents the process that was used for Phase I of the dependability and safety case. Section 3 describes Phase II of the dependability and safety case and transforms a SRM model from the specific to the generic. Section 4 describes how Section 3 can be applied to organizations both in and outside of the space program. Section 5 concludes with a discussion of future work. Section 6 includes the references.

2 Phase I: Overview of Initial Dependability & Safety Case

2.1 Model Safety-critical Behaviors

The NASA IV&V Program conducted a safety case study for a science satellite mission. Requirements validation was conducted on developer provided artifacts and a SRM was developed.

In addition, FMEA and FM artifacts were reviewed for fault conditions that would put the spacecraft in safe-hold. It was observed that the FMEA and FM artifacts largely focused on hardware failures. Those conditions were compared to an IV&V program developed list of safety-critical failure conditions which contained a combination of hardware and software faults. Traceability was created between system and software requirements and faults. From this activity, gaps were documented which helped identify missing safe-hold requirements.

Figure 1 portrays the IV&V analysis process created and followed. Figure 2 is a high-level depiction of the safety case which maps high-level safety requirements and lower-level safety requirements.

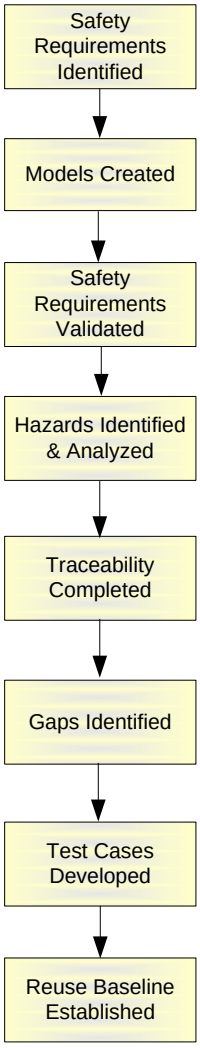


Figure 1 - IV&V Analysis Process [2]

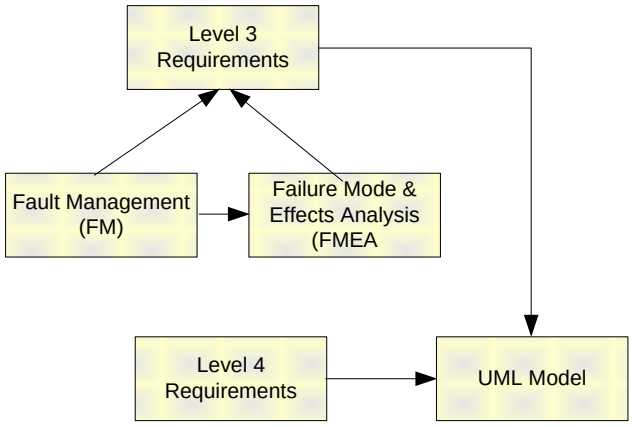


Figure 2 - High-Level Safety Case [2]

The identified gaps indicated that both the mission developer and the IV&V safety-critical failure conditions were incomplete. The IV&V list was updated and discussed with the mission developer who was very responsive to and is acting upon the data provided. Dependability and Safety were enhanced by the creation of a reusable template of artifacts and processes. These items make up the safety case to ensure that hazards are managed. Mission success and spacecraft safety are both improved through contingency hazard management and the resulting failure risk reduction [2].

Throughout this phase it became clear the process identified in Figure 1 was applicable to all of fault management and not just safe-hold and will be incorporated into future work addressed in Phase II.

3 Phase II: From the Specific to the Generic

The initial models and independent list of fault conditions were specific to a single science satellite mission. As an exercise, the independent list of fault conditions was compared to the fault conditions for the Mars Science Laboratory (MSL). Early on, it became obvious that the first mission Phase I IV&V list was only minimally re-usable due to its mission device-specific nature.

This is due to the fact that fault conditions for the Phase I science mission and the IV&V developed fault conditions list were device dependent. Although families of spacecraft may use the same underlying architecture, the subsystem device names are often different. It became clear that the models and the independent list of fault conditions needed to be based on the functionality of a subsystem at the highest level as opposed to the functionality of the device-specific hardware.

When comparing space missions to each other, it was immediately obvious that all missions share many of the same characteristics - regardless of the mission's purpose. All space missions have subsystems that deal with telemetry, command and data handling, guidance navigation and control, 1553 bus, temperatures, voltages etc. Functionality of other missions uses pyrotechnics, robotic rovers and unique experiments. Those subsystems may have differing designs and device names, but the subsystem functionality is the common thread.

Instead of focusing on the specific subsystem device with a specific fault, the focus will be on the functionality of a specific subsystem (Figure 3) with the fault conditions captured at a high and generic level to more easily be reused across other future missions. The generic behavior faults and related hazard management can be detailed later as the knowledge of the subsystem and its needs are discovered. This approach creates a standardized naming convention for dependability and safety cases across multiple system architectures that are reused.

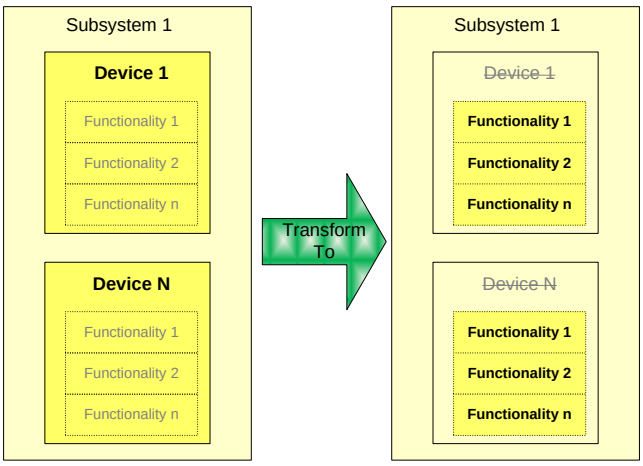


Figure 3 - Focus on functionality

Using a Command and Data Handling (C&DH) example from two distinctly different science missions, we determined the specific device name - Single Board Computer (SBC) from one mission and the specific device name - Flight Computer (FC) from another mission was actually the same device, the RAD750 computers (Figure 4) which we refer to as the main spaceflight computers.

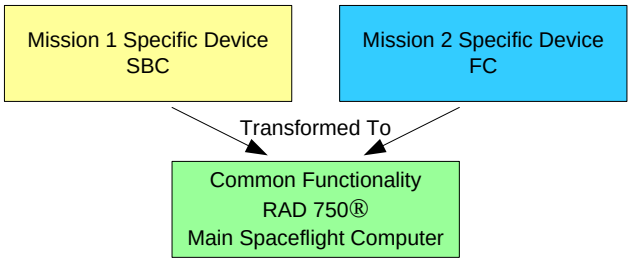


Figure 4 – Identifying common functionality

The RAD750® is commonly used for spaceflight and its functionality is well understood. This allows the re-use of fault conditions from one mission to another regardless of spacecraft family. This doesn't preclude or eliminate the development of, or the need for, additional fault conditions that are unique to a specific mission.

Moving forward, the fault conditions will be divided into three categories:

1. Cruise/orbit
2. Science Experiments
3. Surface operations (interplanetary rovers/landers)

The process in Figure 5 will be used to identify and communicate generic fault condition candidates.

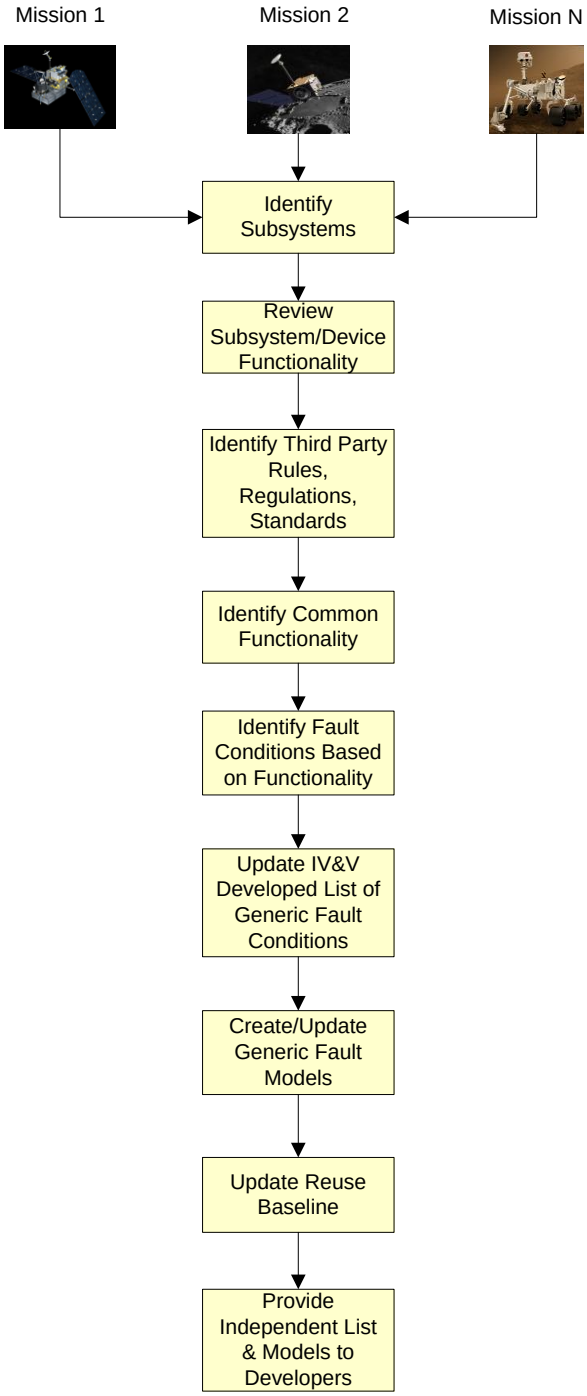


Figure 5 - Process for identifying generic fault conditions

It's expected that the mission developer will use the independent list and models to help create and/or validate their mission-specific fault conditions. The IV&V program also anticipates inputs and updates from the mission developer.

3.1 From the Specific to the Generic: An Example

Figure 6 is an example of an activity diagram which depicts a high-level overview of fault management for a safe-hold event for a specific science mission. Each subsystem is comprised of specific devices in which specific failure(s) would result in a safe-hold event. Due to the proprietary nature of the data, the specific device names have been removed for the purposes of this paper.

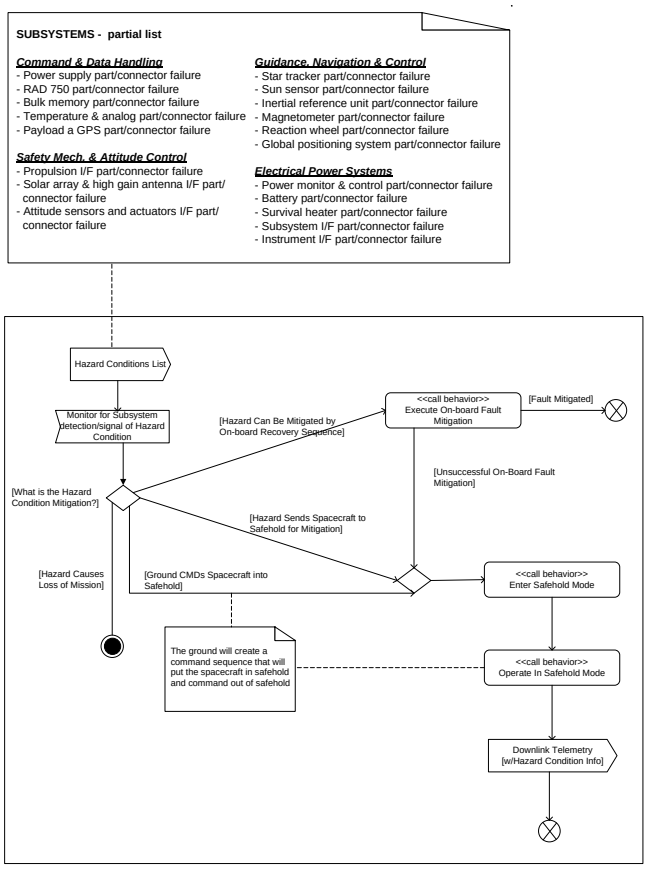


Figure 6 - Mission specific activity diagram for safe-hold fault management [2]

Figure 7 transforms Figure 6 into a generic model of fault management that can be applied to any space mission. Device names were replaced with the functionality of each subsystem which also account for software as well as hardware issues. The activities were modified to include faults of any kind, and are generic enough to be applied to and modified by any mission developer.

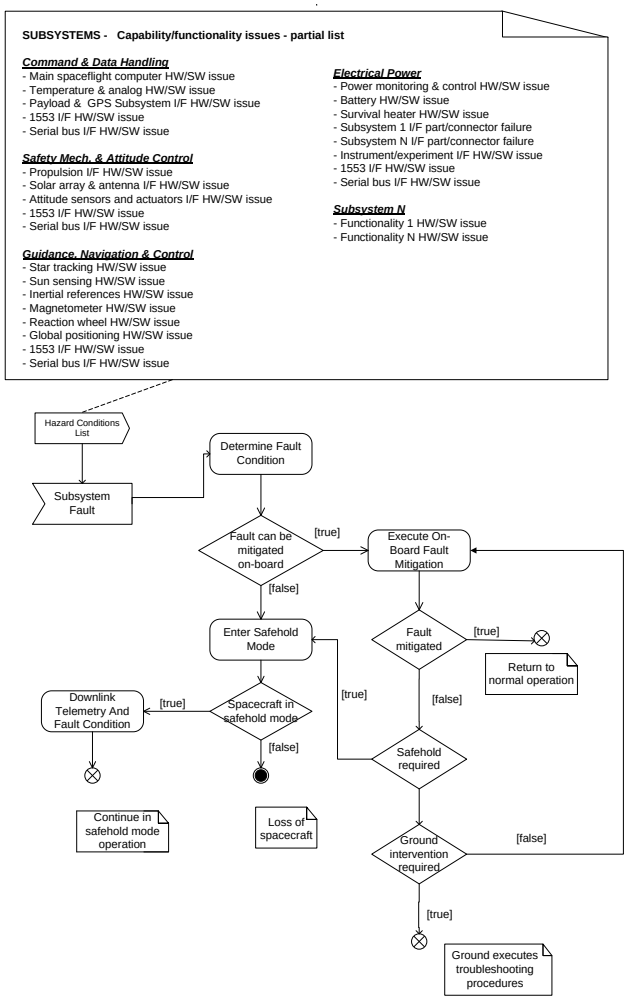


Figure 7 - Activity diagram for generic fault management

Once the subsystem functionality is understood, each unique function is identified and documented. The function is then decomposed into its known and potential hardware and software faults. Using the main computer processor as an example, potential faults include the following:

- Peripheral Component Interconnect (PCI) status register errors
- Excessive accumulation of uncorrectable SDRAM memory errors
- Overcurrent/undercurrent
- Overvoltage/undervoltage
- CPU halt/hung
- Etc

Some of these faults can be further decomposed into more detailed faults. The PCI status register is used to record status information for PCI bus related events [7] and bit 15 can be used to identify a parity error. Instead of referring to the "SBC" or the "FC," both of which are mission specific devices for two different science missions, we propose a reference to the "main spaceflight computer" along with potential faults that the mission developer can then apply to

their specific mission. This allows the mission developer to focus on the functionality and faults of a subsystem as opposed to being hindered by device names.

The benefits of this concept are multi-fold:

1. A pre-defined list of fault conditions provides a starting point for fault identification for any mission – doesn't recreate the wheel
2. A pre-defined list of fault conditions can be compared to existing fault conditions to identify gaps in both source requirements and elucidated faults – are the bases covered?
3. A pre-defined list of fault conditions increases the mission success and spacecraft safety probabilities through comprehensive contingency hazard management – is the list a superset of conditions? Is the hazard management adequate?

4 Applying Phase II to Your Project

Modifying Figure 5, your organization can apply similar fault management techniques even if your projects do not have the SoS complexity (Figure 8).

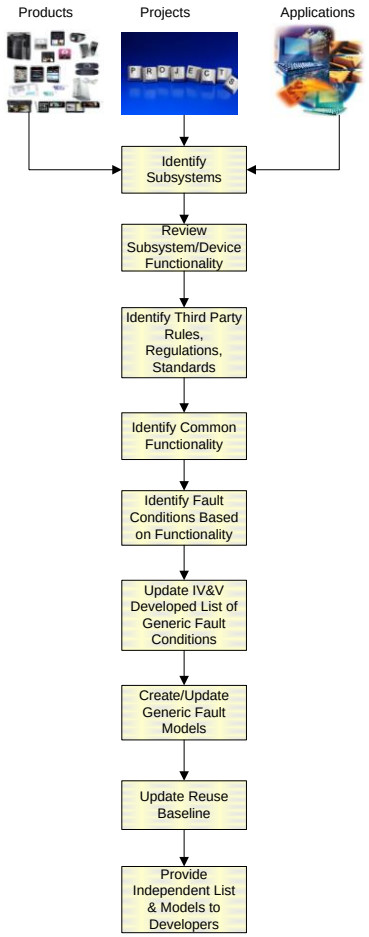


Figure 8 - Applying Phase II to your project

Replace the space mission examples with your system information. Decompose the system into subsystems (Figure 3) with a focus on subsystem functionality. Instead of reviewing device functionality, replace that with your own projects, programs or applications. When you identify common functionality, add your lessons learned from previous projects. These products are specific to your business and establish a reusable baseline of artifacts for use on multiple projects, where they become a library of analysis models.

Don't think spaceflight – think your business (Figure 9):

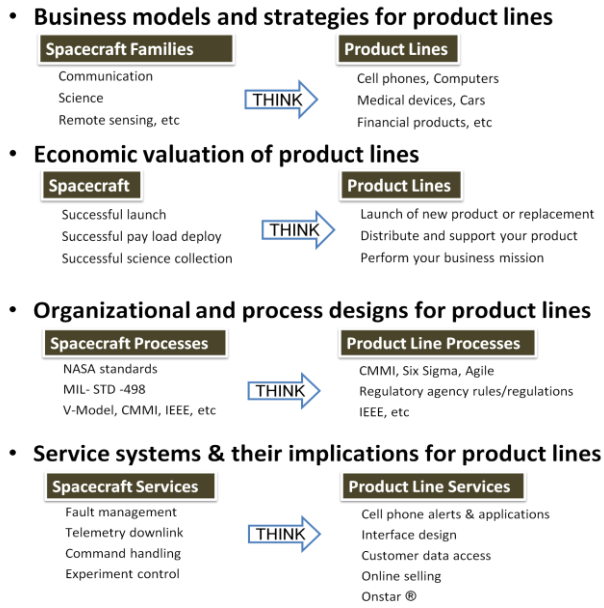


Figure 9 - Thinking about the same thing in a different way

This will allow your organization to efficiently identify fault conditions which accelerates your system and software development. This IV&V process approach ensures software and system quality. Using the IV&V three questions ensures that you have considered the special conditions necessary to verify that the software is properly implemented.

5 Conclusion

In this paper we present a proactive approach to identifying reusable fault conditions based on the functionality of a SoS instead of identifying fault conditions based solely on specific architecture implementation.

Phase II introduces a new way to independently validate software safety requirements, via the comparison of the FMEA, FM and FFA artifacts against the IV&V team's own list of fault conditions. This helps the mission developer ensure they have identified the correct fault conditions and will help the IV&V program keep their independent list current. This will also help the mission

developer identify missing requirements (shown as gaps by IV&V, through this developer interaction process). This will allow the IV&V program and the mission developer to do the following:

- Build a foundation for dependability and safety that is reusable
- Identify room for improvement in the IV&V program's processes that extends to all projects
- Identify missing fault condition initiating failure events, which result in hazards
- Identify missing safety requirements
- Contribute to the goodness of any mission

FM requirements are necessary for all NASA space missions. All of the mission's are a SoS comprised of a combination of legacy systems and new development [2]. Phase II has started with the creation of generic fault conditions for cruise/orbit portions of a mission.

Phase III will continue with fault conditions for experiments and Phase IV will continue with fault conditions for surface operations for planetary robotic missions.

Using the concepts provided in this paper, any organization can generalize for reuse their projects hazard controls using the process identified in Figure 3.

A future goal of this study is to build not only generic fault management monitors and controls by technology type, but to also provide automated models with technology to test, using Application Program Interfaces (APIs). This enables automation of the design and implementation validation and verification process. These modifications to manual analysis contribute to dependability, safety, availability, reliability performance, maintainability and security. It accelerates the attainment of these goals and other safety objectives. Accomplishing the approach outlined in this paper contributes to the reusability of software in general, as well as the IV&V processes used to ensure mission success.

6 References

[1] US Government Accountability Office, NASA: Assessments of Selected Large-Scale Projects, Report Number GAO-10-227SP, February 1, 2010. Accessed on 29, March 2010: http://www.gao.gov/products/GAO-10-227SP.

[2] S. Driskell, J. Murphy, J.B. Michael and M. Shing. "Independent Validation of Software Safety Requirements for Systems of Systems," *Proc. 2010 IEEE International Conference on System of Systems Engineering*, Loughborough University, UK, 22-24 June 2010.

[3] NASA Software Safety Standard (w/Change 1 dated 7/08/04), NASA-STD-8719.13B, March 12, 2008. Accessed on 14, November 2010: http://www.hq.nasa.gov/office/codeq/doctree/871913.htm.

[4] NASA General Safety Program Requirements (w/Change dated 7/20/09), NPR 8715.3C, March 12, 2008. Accessed on 14, November 2010: http://nodis3.gsfc.nasa.gov/npg_img/N_PR_8715_003C_/N_PR_8715_003C_.pdf.

[5] W. Gibbs, "Software's Chronic Crisis," *Scientific American*, pp. 86, Sep. 1994.

[6] NASA Software Safety Guidebook (w/Change 1 dated 7/08/04), NASA-GB-8719.13, March 31, 2004. Accessed on 14, November 2010: http://www.hq.nasa.gov/office/codeq/doctree/871913.htm

[7] PCI Local Bus specification, Revision 2.2, December 18, 1998. Accessed on 19, November 2010: http://www.ece.mtu.edu/faculty/btdavis/courses/mtu_ee3173_f04/papers/PCI_2d2.pdf.

[8] "NASA IV&V Vision and Mission." NASA IV&V Facility, accessed 9, December 2010, http://www.nasa.gov/centers/ivv/about/visionmission.html.

[9] author must maintain a record for the use of exemption 125.4(b)(13) for five (5) years from the date the paper is placed in the public domain 17 DEC 2010. - 22 CFR 125.4(b)(13) applicable log (TCO-10-1048): Expires 17 DEC 2015.

A design of software metric tool for improving reliability and usability

Won Shin[1], Tae-Wan Kim[2], Doo-Hyun Kim[3, *] and Chun-Hyon Chang[1]
[1] Department of Computer Engineering, University of Konkuk, Seoul, Korea
[2] Department of Electrical Engineering, University of Myongji, Gyeonggido, Korea
[3] School of Internet and Multimedia Engineering, University of Konkuk, Seoul, Korea

Abstract - *Software Metric can provide essential information used for software development and maintenance processes. There have been also lots of researches on software metric tool helping users to use the software metric easily. However, many developers and project managers have suffered difficulties in using the tools. They could not trust on the result of metrics since it was needed to master the usage of the metrics in order to elicit a result from the tools. To overcome these difficulties, this paper suggests a parametric software metric tool (PSMT). PSMT came from the idea of parametric analysis technique and follows 15 requirements for making a software metric tool, hence, can provide appropriate information according to user's goal as well as user can use it easily. Moreover, it can be utilized for analyzing all kinds of metrics by simply adding them.*

Keywords: Software Metrics, Parametric Analysis, Metrics Tool, Static Analysis

1 Introduction

Many of the best software developers tend to measure characteristics of the software to get a priori experiences on the consistency and completeness of the requirements, the quality of the design, and the readiness of the codes for tests. Effective project managers want to measure the attributes of a process and product for being able to forecast the time when the software shall be ready for delivery and the budget shall be exceeded [9]. Thus, both of the developers and project managers sufficiently acknowledge the need of software metric tools. However, many of them insist on the difficulties of their usages as well as they do not believe the result of metrics because it requires to understand how to use it and the tools of the metrics do not show any information on the course to reach such result. Moreover, the results are different from each other due to their own different criteria [1, 10]. To resolve these problems, this paper suggests parametric software metric tool (PSMT) which is based on the idea of parametric WCET analysis [2, 4, 6]. The idea is to make parametric analyzer to produce a formula instead of a constant for unknown values, since it is impossible to obtain proper values ahead by analyzing program codes only. The unknown values are called parameters in the parametric analysis. A user gets a result throughout inputting the values of parameters. The advantages of this way are to improve flexibility and reliability of tools because users can know the analysis process of a metric tool because the definition formulas and attributes are made by themselves, and the metric tool can be changed by the user's goal. The goal is the quantitative value that the user wants to know through the analysis on their software. In order to apply the parametric analysis technique to software metric tool, we first categorize software attributes in detail and then generate the formula using them. All types of attributes and formulas are saved as resources in storage. Additionally, the usability of a tool such as visualization, customizing functions and so on is considered.

The rest of the paper is organized as follows. Section 2 describes various software metric tools and their problems Section 3, then, presents a design of the parametric software metric tool. Finally, Section 4 discusses on the effectiveness of PSMT and concludes with a suggestion on future works.

2 Software metric tools

2.1 Kinds of Software metric tools

Software Metrics have to be changed by the programming language, software development process, users and usage goal. On the other hand, software metrics are generated throughout choosing them. This means that metrics have a limitation of usages since the metrics can be just applied in its own environment.

There have been lots of researches on the metric tools which can help to use the software metric easily. RSM [11] supports the phases of software metric usage. Each of phases is a step for making a result of the metric. A user would finish all of the phases and then gets a result of the metric. It is very simple to use it because the user just chooses his software and clicks some buttons. The Understand [13] and the Metrics [8] support various visualization techniques such as graph, bar and so on. Those help users to understand and realize the result easily. In other researches, they tried to compare those

* Corresponding author.

tools and analyze the differences and characteristics of those tools as well [7].

2.2 Some problems of software metric tools

Previous process of software metrics are largely divided into two modules. The program analyzer firstly analyzes program codes. Secondly, it generates the static information which contains some values of software attributes for evaluating software metric and extra data such as control flow graph, program description and so on. The metric evaluator calculates and exports various document types to use the result of metric. In this process, the user can not recognize the internal procedures on reaching such result, neither can do on deciding a scope of software attribute. After all, users consequently feel difficulties in using these kinds of tools and do not trust on the result derived from metrics.

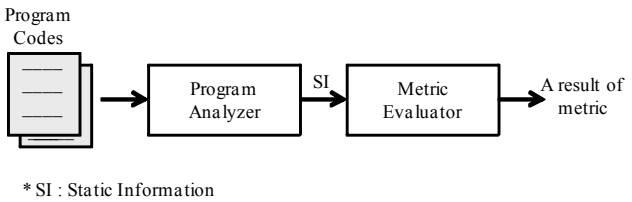


Figure 1 previous process of software metric

There have been several researches to overcome those difficulties and to improve reliability of the result from metrics. Cem [3] suggested a new method to compute a precise result of metrics. It first surveys user's goal, language, and etc, and then makes a metric based on the survey results. The metric would make an appropriate result that the user wanted. However, it is absolutely difficult to make a questionnaire and it is also needs to pay a close attention because even a single answer to the questionnaire can affects the metric and the result from it. Moreover, it is costly in time to make a metric according to the survey results. Fenton [5] tried to classify software attributes in detail because software metric was ambiguous in scope and definition. For instance, Line of Code (LOC) is one of the most popular software attributes. But, it is ambiguous whether it contains comment lines or not. Some metrics may contain it others not. Consequently, there still exist problems about reliability of tool in spite of their efforts. Therefore, it is absolutely needed to consider the ways of improving reliability of the result when making a software metric tool.

2.3 Conceptual Requirements for metrics

Hashem [7] suggested conceptual requirements for metrics. They also compared and analyzed several tools according to their requirements. However, there were no tools satisfying their requirements, thus it is necessary to invent a new metric tool reflecting all the requirements.

In Table 1, the number 2, 4, and 5 are closely related with the problems of metric which we already explained in 2.2. And the other requirements are concerned with the usability of metric tool. Consequently, it has become necessary to resolve the reliability problem and to consider the usability in making a metric tool.

3 A design of parametric software metric tool

The reasons that users do not trust on the result of metric are considered that they have no permission to change any parts of tools and the tools do not show a process of analyzing metric. To make thing worse, the tools are not flexible enough. In order to improve the flexibility of tools, we apply a parametric analysis technique to the software metric tool. The parametric analysis technique generates formulas instead of constant as a result [2, 4, 6]. The formula contains unknown variables, called parameters, which cannot be given by analyzing program codes only. Instead, users can input a value of parameter by using annotation language, file, and console input. As a result, a constant can be computed according to the given values for the parameters. One of its advantages is to improve flexibility because the result can be changed by the value of parameter decided by the user.

For applying the technique, it is needed to define what to use instead of parameter and what to generate as a result. Software attribute on the metric is used as a parameter in the formula and the software quality is the result of tool instead of the worst case execution time (WCET) which is generated by parametric WCET analyzer. We will explain about it in detail in section 3.2.

3.1 Architecture of PSMT

The design of PSMT is based on 15 kinds of considerations showed in Table 1. In order for applying the parametric technique, some considerations such as customizing, extension and so on are included automatically.

The program analyzer generates static information derived from a program codes and the metric evaluator makes metric information according to the generated static information. The metric information consists of combined metric with static information, differences among results, and history of a result, and etc. Those are used for proper visualization and exported by the metric reporter.

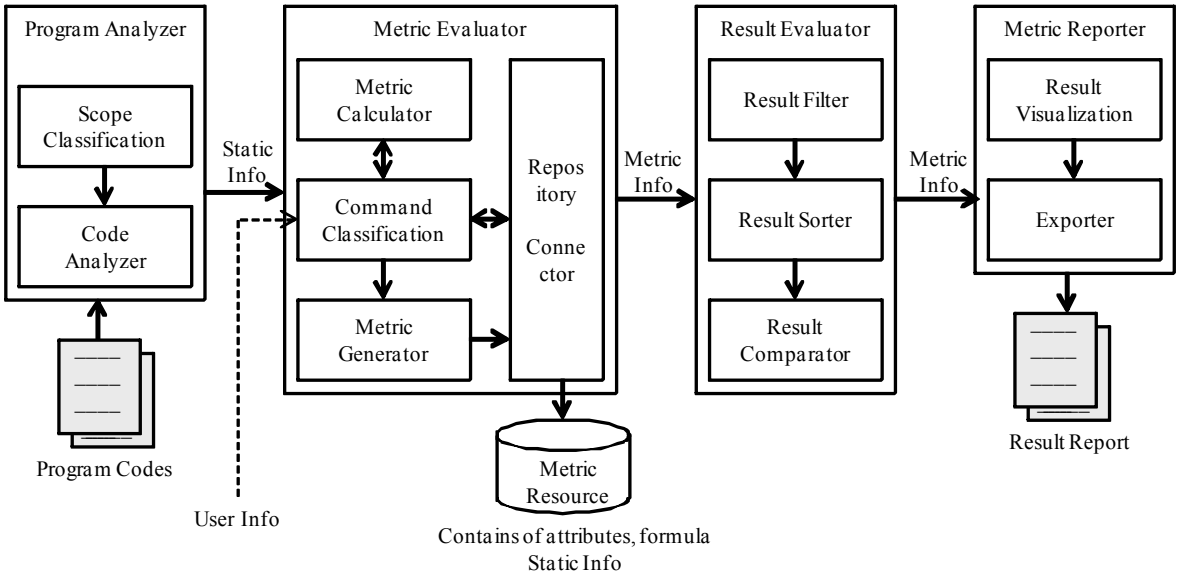


Figure 2 Architecture of PSMT

3.2 The metric evaluator of PSMT

The metric evaluator is the most important part of PSMT. The metric evaluator also manages variety of information based on the user information, which is to reflect user's goal or opinion. The user information consists of formulas, software attributes, filters, metric set and so on.

The formula represents relationships among software attributes shown in Fig 3. There are two metrics, M1 and M2, and we suppose that they have the same role. M1 is composed of A1, A3 and plus operator. M2 is made up of A2, A4 and minus operator. Those would generate different results respectively. It shows that a result of metric will be changed by the formula and PSMT can generate more suitable result than previous metric tools if a user determines software attributes and makes a formula.

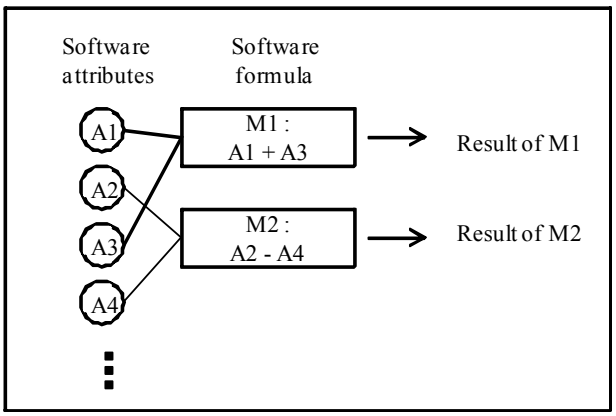


Figure 3 Conceptual relationships between attributes and metrics

Filter is the boundary of attributes and it can be used to compare results of metrics. For example, if a user needs to get LOC of their code which is less than 1,000. In this case, the boundary is less than 1000 and the attributes is LOC.

The metric set is organized of metric set name, several kinds of the formulas, description, and filters. Default metric sets are needed for elementary users and it can be modified by advanced users.

3.3 Modules of PSMT

The metric evaluator is related to resolve the flexibility and to improve the reliability. The other modules are concerned with the usability support. Accordingly, they just consider functionalities on supporting information easily and improving readability. Table 2 shows how each module is associated with requirements in Table 1.

4 Conclusions

The previous software metric had problems related to the flexibility and it was also needed to improve the usability in software metric tools. Users, therefore, have been feeling difficulty to use the tools. To overcome those difficulties, we, in this paper, proposed a new method called PSMT, where we not only added parametric analysis technique to software metric for flexibility, but also designed metric tool considering some requirements suggested in [7]. Consequently, PSMT could provide appropriate information to users as well as it helped users to understand software metric through making their formula. They could understand the meaning of the result from metric and they also could realize how and what to improve for their software. Moreover,

expanding software metric can be done by simply adding a new formula into storage.

In order to use PSMT, however, the user has to understand the concept of software metric and make a formula according to his goal. Although it could be a burden due to using PSMT, the knowledge of software metric is necessary after all because it is known as commonsense in using software metric tools. Therefore, PSMT requires minimum burden, thus, it will be better to use PSMT than other metric tools.

In the future, we will implement the PSMT and will carry out a survey targeting software developers.

Table 1. Conceptual Requirements for metrics

No	Considerations	Description
1	Metrics coverage	Should provide the maximum possible number of different metrics of the supported programming paradigm.
2	Metrics providing	Should support explanations on how the metric is calculated, and how the results can be used.
3	Metrics suites	Should a default subset of all available metrics be active for non- or less experienced developers.
4	Metrics Customizing	Should be able to change the settings if necessary.
5	Metrics Extending	Should also allow the definition of self defined metrics.
6	Metrics feedback	Should be able to give tips on how to interpret the results and on what to do to improve them.
7	Metrics filtering	Should provide value filters and component filtering.
8	Sorting results	When viewing the output of metrics, it should be able to compare and organize the items in the result table.
9	Metrics visualization	Metrics results should be viewed as graphical output and metrics should use different kinds of visualization of the measurement data such as Bar chart, Distribution Graph, Histogram, Scatter plot and Pareto chart and etc.
10	Saving and loading metrics results	Should be able to save the results and later view the results table independently of the project.
11	Comparing metrics results	Should help developers to control his work by comparing projects or by comparing changes in a project over time.
12	Printing results	Should preview and print results either by printing the table or by printing graphs
13	Exporting results	Should be possible to generate a report in separate file in various formats such as text, HTML, csv and etc.
14	Copying metrics results to the clipboard	Should be able to copy results from the Code Metrics.
15	Metrics verification	The results of metrics analysis are tightly connected with source code.

Table 2. Modules of PSMT

Module Name	Description	Number of requirements
Scope Classification	Classify program codes according to criteria for classification.	7
Metric Calculator	Compute a result by using formula and values of parameters.	2
Metric Generator	Define software attributes and formula according to user info.	1,3,4,5,6
Result Filter	Classify a result of metric according to maximum or minimum boundary.	7
Result Sorter	The result and history of the result are sorted by user order.	8
Result Comparator	Analyze differences between current result and previous results.	11
Result Visualization	A result of metric is visualized as a table, chart and so on.	9,10,12
Exporter	A result of metric is exported to file, clipboard and so on.	13, 14

5 Acknowledgments

This research was supported by R&DB Support Center of Seoul Development Institute, Korea, under Seoul R&BD Program (ST100107) and was also supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency)" (NIPA-2011-C1090-1131-0003)

6 References

[1] Amandeep Kaur, Satwinder Singh. " Empirical Analysis of CK & MOOD Metric Suit". Int. Journal of Innovation, Management and Technology, Vol 1. No 5., pp. 447-452, 2010

[2] B. Lisper. "Fully automatic, parametric worst-case execution time analysis". 3rd International Workshop on Worst-Case Execution Time Analysis, pp. 99-102, 2003

[3] Cem Kaner, Walter P. Bond. "Software engineering metrics : What do they measure and how do we know?". In 10th Int. Software Metrics Symposium, IEEE CS Press, 2004

[4] Emilio Vivancos, Chistopher Healy, Frank Mueller, David Whalley. "Parametric Timing Analysis". Workshop on Language, Compilers, and Tools for Embedded Systems, Vol 36, Issue 8, pp. 88-93, ACM Press, 2001

[5] Feton. "Quality Assurance and Metrics". http://www.eecs.qmul.ac.uk/~norman/papers/qa_metrics_article/index_qa_met.htm

[6] Guillem Bernat, Alan Burns. "An Approach To Symbolic Worst-Case Execution Time Analysis". Proc. In 25th IFAC Workshop on Real-Time Programming, 2000

[7] Hashem Yazbek. "A concept of quality assurance for metrics in CASE-tools". ACM SIGSOFT Software Engineering Notes, Vol 35, Issue 5, pp. 1-7, Sep 2010

[8] Metrics. http://metrics.sourceforge.net/

[9] Norman E. Fenton, Shari Lawrence pfleeger. "Software Metrics: A Rigorous and Practical Approach Second Edition". In: PWS Publishing Co., 1997

[10] Paramvir Singh, Hardeep Singh. "DynaMetrics: A Runtime Metric-Based Analysis Tool for Object-Oriented Software Systems". SIGSOFT Software Engineering Notes, Vol 33, Issue 6, pp. 1-6, Nov 2008

[11] Resource Standard Metrics. http://msquaredtechnologies.com/

[12] Rudiger Lincke, Jonas Lundberg. "Comparing software metrics tools". Proc. of the 2008 international symposium on Software testing and analysis (ISSTA'08), pp. 131-142, 2008

[13] Understand. http://www.scitools.com/index.php

SESSION

SOFTWARE PROJECT MANAGEMENT + EDUCATION

Chair(s)

TBA

Evaluation of Business Software Systems Development and Enhancement Projects Effectiveness and Economic Efficiency on the basis of Functional Size Measurement

Beata Czarnacka-Chrobot
Department of Business Informatics, Warsaw School of Economics, Warsaw, Poland, bczarn@sgh.waw.pl

Abstract - *Each rational investment decision, also that made by client with regard to the Business Software Systems (BSS) Development and Enhancement Projects (D&EP), should meet two measurable criteria: of effectiveness and of economic efficiency. In the case of BSS D&EP the assumption concerning measurability of these criteria is often treated as controversial. Thus the paper aims at proving from theoretical perspective the capabilities of using the so-called concept of BSS D&EP Functional Assessment (FA), proposed by the author and already verified in practice, based on Functional Size Measurement (FSM) concept and methods, in the area of ex ante and ex post quantitative evaluation of these two criteria. By linking the FSM issues with economic aspects it may contribute to better understanding of the FSM importance, still being underestimated by business managers. Meanwhile, BSS D&EP FA can constitute the basis for rational decisions not only for BSS providers, but also for clients. These issues classify into economics problems of Software Engineering Research and Practice (SERP).*

Keywords: rational investment decision, functional size measurement, functional assessment, business software systems development and enhancement projects effectiveness and economic efficiency

1 Introduction and related work

Each rational investment decision should meet three basic criteria [1], which in the context of Business Software Systems (BSS) Development and Enhancement Projects (D&EP) should be interpreted as fallows:

- Criterion of economic efficiency, meaning that the decision should benefit to the maximisation of the relationship between the effects to be gained as a result of project execution and the costs being estimated for the project.
- Criterion of effectiveness, meaning that such decision should contribute to achieving the assumed result, in the case of BSS D&EP usually being considered as delivering product meeting client's requirements with regard to functions and features without budget and time overruns.
- Criterion of consistency, which means that the project should comply with the environment (economic, organisational, legal and cultural) – unlike the above two criteria, this criterion is not subject to quantitative assessment therefore it is skipped in this paper.

Generally speaking, in the case of economic efficiency evaluation, effects are compared against costs necessary to achieve these effects while in the case of effectiveness evaluation these are only results that are of significance. Thus, economic efficiency (*Ef*) is measured by relating total effects (*Efec*) to total costs (*C*), most often as:

$$Ef = Efec / C. \quad (1)$$

Meanwhile, effectiveness (*Efs*) is measured by the ratio of the achieved result (*Ra*) to the assumed result (*Ras*), which is being conveniently expressed as a percentage:

$$Efs = (Ra / Ras) \times 100\%. \quad (2)$$

Both economic efficiency criterion as well as effectiveness criterion is based on the obvious assumption that the effects, costs and results are measurable. However, in the case of BSS D&EP this assumption is often treated as controversial. Numerous studies indicate that evaluation of BSS D&EP efficiency is made relatively rarely while fundamental reason for this *status quo* are difficulties related to identification, and most of all quantitative expression, of benefits resulting from the execution of such projects (see e.g., [2], [3], [4], [5], [6]). These studies reveal that difficulties related to identification and quantitative expression of BSS D&EP costs too are of significance, which also is of importance to the evaluation of their effectiveness.

Key conclusions coming from the above mentioned studies have been confirmed also by the results of studies carried out by the author of this paper in two research cycles among Polish dedicated BSS providers [7]. They revealed that at the turn of the years 2005/2006 the results obtained with the use of the effort estimation methods, employed only by approx. 45% of the respondents, were designed for estimating BSS D&EP costs and time frame while relatively rarely they were used to estimate economic efficiency – such use of these methods was indicated by only 25% of those using effort estimation methods. Heads of IT departments in Polish companies, for which BSS D&EP are executed, still explain the sporadically required calculation of this type of investments efficiency mostly by the necessity to undertake them – most often due to the fact that without such solutions they lack possibility to match competition from foreign companies, as well as to match foreign business partners requirements. While Polish public administration institutions in practice still do not see the need for the BSS D&EP economic efficiency evaluation, in most cases as an argument giving the non-economic purposes of systems being implemented in this type of organisations. On the other hand, at the turn of the years 2008/2009 the results obtained with the use of the BSS D&EP effort estimation methods (approx. 53% of BSS providers surveyed in this cycle declared they commonly employed such methods) were more often used to estimate efficiency: there was an increase to approx. 36% of those using effort estimation methods. This applies to internal IT departments of Polish companies yet still it does not

comprise public administration institutions. This increase may be explained first of all by stronger care about financial means in the times of recession, however it still leaves a lot to be desired.

What's more, majority of BSS D&EP fail to meet criteria of their effectiveness, what leads to the substantial financial losses, on a worldwide scale estimated to be hundreds of billions of dollars yearly. As indicated by the results of Standish Group analyses success rate for application D&EP has never gone beyond 35%, while currently products delivered as a result of nearly 45% of them lack on average 32% of the required functions and features, the estimated project budget is exceeded by approx. 55% on average and the planned time – by nearly 80% on average [8]. In the case of new application development the success rate is only 4% [9]. Meanwhile, analyses by T.C. Jones plainly indicate that those software D&EP, which are aimed at delivery of BSS, have the lowest chance to succeed [10]. It was proved also by the Panorama Consulting Group, which revealed that merely 7% of the surveyed ERP (Enterprise Resource Planning) systems projects were accomplished as planned [11].

All above presented results unequivocally implies a significant need to rationalize investment decisions made with regard to BSS D&EP. For each BSS D&EP there are functional goals being set. Thus if one assumes that fundamental benefit for a client coming from the execution of such project is functionality of BSS or increase in functionality it brings, it may then justify the necessary investment costs. What allows for making this assumption is a specificity of the considered types of projects, which indicates that BSS D&EP product functionality is an attribute of priority significance to a client, deciding on the possibility of business processes execution therefore determining the value of BSS. What is more, software product functionality may be expressed quantitatively – with the use of the so-called product functional size, for which are meant the concept and the methods of Functional Size Measurement (FSM), having been recently standardized by the ISO/IEC (see [12], [13], [14], [15], [16], [17]), thus constituting rational basis for determining BSS D&EP key attributes (work effort, cost and duration). The product functional size is defined as „size of the software derived by quantifying the Functional User Requirements", while functional user requirements (FUR) stand for the „sub-set of the User Requirements describing what the software does, in terms of tasks and services" [12, Part 1].

The FSM concept and methods constitutes basis of the southernSCOPE [18] and northernSCOPE [19] methodologies supporting the management of BSS D&EP functional scope, i.e., scope measured on the basis of functional size of their product. Concurrently with the above methodologies the author of this paper proposed and verified in practice her own concept, designed for the functional assessment of BSS D&EP. As this paper aims at proving from *theoretical* perspective the capabilities of using this approach in the context of *ex ante* and *ex post* quantitative evaluation of BSS D&EP effectiveness and economic efficiency, the short presentation of the functional assessment concept appears vital here as well. It should be mentioned that due to the limited volume of the paper it will only feature conclusions coming from the *practical* verification of functional assessment concept – this verification is widely presented in [20] and [21].

2 Functional Assessment of BSS D&EP

The FSM capabilities can be used for the assessment of BSS D&EP from a functional – that is from the most important to a client – point of view. *Ex ante* and *ex post* assessment of such projects made on the basis of their products' FSM will be then called Functional Assessment (FA). Key attributes of FA include: product functional size (*FS*), work effort which needs to be spent on *FS* development/enhancement (*E*), and functional productivity (*P*) understood as the ratio of product functional size to the work effort on *FS* development/enhancement (*FS/E*), or – being inversion of functional productivity – work effort necessary to achieve functionality unit ($E(fu)=E/FS$) that determines work cost per *FS* unit ($C(fu)$).

In order for BSS D&EP to be considered as complying with the criteria of FA, its ***r***equired (*FSr*, *Er*, *Pr*), ***o***ffered (*FSo*, *Eo*, *Po*) and ***re***alized (*FSre*, *Ere*, *Pre*) functional attributes should meet the following conditions:

1. Product functional size – both required by a client (*FSr*) as well as offered (*FSo*) and realized (*FSre*) by a provider – must be within the range allowed for *FSr*, i.e., [*FS*min, *FS*max], where: *FS*min – stands for minimum while *FS*max – stands for maximum required functional size. Defining of *FS*max results from the fact that only about 20% of functions and features specified ever get used [8]. Thus delineating *FS*max reduces the risk of delivering needless functionality.
2. Work effort – both expected by a client (*Er*) as well as offered (*Eo*) and realized (*Ere*) by a provider – must be within the range allowed for *Er*, i.e., [*E*min, *E*max], where: *E*min – stands for minimum while *E*max – stands for maximum effort expected by a client. *E*min should not be lower than the effort enabling for delivering minimum required functional size (*FS*min).
3. Functional productivity – both required by a client (*Pr*) as well as offered (*Po*) and realized (*Pre*) by a provider – must be within the range allowed for *Pr*, i.e., [*P*min, *P*max], where: *P*min – stands for minimum while *P*max – stands for maximum productivity required by a client. Having *P*max defined is useful for rational provider offer selection, i.e., from the point of view of limiting the risk of choosing the offer where the productivity would be defined as overstated value. Since such situation would mean that in fact the effort per functionality unit (i.e., function point) is likely to be exceeded, what would entail the risk of delivering product having functional size lower than the allowed one as the provider would be probably trying not to go over the offered effort. In addition, delineating *P*max is conducive to the increased probability of delivering product of sufficient quality.

Allowed minimum and maximum values of functional attributes (*FS*min, *FS*max, *E*min, *E*max, *P*min, *P*max) depend on the development stage and thus on the FA stage. They take into account maximum allowed (in accordance with the rules of FSM) estimation error: at the analysis stage estimation error up to ±30% is allowed

while in the case the detailed FUR are already known, estimation error should not exceed ±10%. Attributes required by client (*FSr, Er, Pr*) are being corrected by this error – minimum and maximum values of functional attributes allowed at given stage are thus calculated.

The verification of FA approach, presented widely in [21], showed that fulfilling these conditions ensures:

- Rationality of client requirements with regard to the FA attributes.
- Conformity of the potential provider offers with rational client requirements concerning FA attributes.
- Conformity of the realized project with rational client requirements concerning FA attributes.

Advantage of the FA concept over southernSCOPE and northernSCOPE methodologies, proved and explained in detail in [20], results from the fact of the concept adopting two significant assumptions, not being explicitly specified in these methodologies, namely:

- Need to apply upper bounds of the allowed tolerance intervals for required, offered and realized functional size and functional productivity and lower bounds for work effort.
- Need to employ at least two stages of estimation: first one for proper assessment of the investment decision rationality while second stage – in order to choose suitable product provider.

Therefore, comparing to these methodologies, usage of the FA concept reduces the risk of choosing inappropriate provider as well as the risk of lowered *ex ante* and overstated *ex post* product pricing, and consequently, it reduces the chance of failing to deliver required functionality and/or product of insufficient quality.

3 Using Functional Assessment to the BSS D&EP effectiveness evaluation

What appears to be of particular importance in view of problems concerning effective execution of BSS D&EP that may be found in practice (see section 1) is the possibility to specify quantitative criteria allowing for the evaluation of compatibility degree of the submitted providers' offers and the executed project with client requirements. Delineation of FA attributes allows to define indicators of both offered and realized effectiveness with regard to such attributes. Hence defining (see formula (2)):

- Compatibility degree of *FSo* (*CDFSo*), where:

$$CDFSo = (FSo \,/\, FS\max) \times 100\%,$$

  allows to express quantitatively the conformity level of functionality offered by provider with optimum functionality for a client.
- Compatibility degree of *Eo* (*CDEo*), where:

$$CDEo = (E\min \,/\, Eo) \times 100\%,$$

  allows to express quantitatively the conformity level of work effort offered by provider with work effort being optimum to a client.
- Compatibility degree of *Po* (*CDPo*), where:

$$CDPo = (Po \,/\, P\max) \times 100\%,$$

  allows to express quantitatively the conformity level of the offered functional productivity with functional productivity being optimum to a client.
- Compatibility degree of *FSre* (*CDFSre*), where:

$$CDFSre = (FSre \,/ FS\max) \times 100\%,$$

  allows to express quantitatively the conformity level of functionality realized by provider with optimum functionality for a client.
- Compatibility degree of *Ere* (*CDEre*), where:

$$CDEre = (E\min \,/\, Ere) \times 100\%,$$

  allows to express quantitatively the conformity level of work effort realized by provider with that being optimum to a client.
- Compatibility degree of *Pre* (*CDPre*), where:

$$CDPre = (Pre \,/\, P\max) \times 100\%,$$

  allows to express quantitatively the conformity level of the realized functional productivity with functional productivity being optimum to a client.

Fulfilling the functional assessment conditions presented in section 2 means that the following conditions for the *offered effectiveness* indicators are also met:

1. *CDFSo* is within the allowed range, that is:

$$CDFSo(\min) \le CDFSo \le CDFSo(\max),$$

   where:
   - *CDFSo*(min) – minimum accepted compatibility degree of functionality offered by provider (*FSo* = *FS*min), meaning sufficient *expected* effectiveness of functionality requirements execution, that is:

$$CDFSo(\min) = (FS\min \,/\, FS\max) \times 100\%,$$

   - *CDFSo*(max) – maximum accepted compatibility degree of functionality offered by provider (*FSo* = *FS*max), that is:

$$CDFSo(\max) = (FS\max \,/\, FS\max) \times 100\% = 100\%.$$

2. *CDEo* is within the allowed range, that is:

$$CDEo(\min) \le CDEo \le CDEo(\max),$$

   where:
   - *CDEo*(min) – minimum accepted compatibility degree of work effort offered by provider (*Eo* = *E*max), meaning sufficient *expected* effectiveness of work effort requirements execution, that is:

$$CDEo(\min) = (E\min \,/\, E\max) \times 100\%,$$

   - *CDEo*(max) – maximum accepted compatibility degree of work effort offered by provider (*Eo* = *E*min), that is:

$$CDEo(\max) = (E\min \,/\, E\min) \times 100\% = 100\%.$$

3. *CDPo* is within the allowed range, that is:

$$CDPo(\min) \le CDPo \le CDPo(\max),$$

   where:
   - *CDPo*(min) – minimum accepted compatibility degree of functional productivity offered by provider (*Po* = *P*min), meaning sufficient *expected* effectiveness of functional productivity requirements execution, that is:

$$CDPo(\min) = (P\min \,/\, P\max) \times 100\%,$$

   - *CDPo*(max) – maximum accepted compatibility degree of functional productivity offered by provider (*Po* = *P*max), that is:

$$CDPo(\max) = (P\max \,/\, P\max) \times 100\% = 100\%.$$

Among offers of BSS providers meeting the above conditions, this is the offer with the highest *CDPo* that is best suited to client's requirements with regard to the criteria of FA – since it stands for the highest offered

allowed functional productivity, or the lowest offered allowed work effort per functionality unit, which decides on unit work cost measured with regard to the functional size unit. This cost, along with the required functional size (*FSr*), should constitute basis for the formal *ex ante* pricing of project product (for more details see [21]).

Meeting the functional assessment conditions presented in section 2 means that the following conditions for the *realized effectiveness* indicators are also fulfilled:

1. *CDFSre* is within the allowed range, that is:

$$CDFSre(\min) \le CDFSre \le CDFSre(\max),$$

where:

- $CDFSre(\min)$ – minimum accepted compatibility degree of functionality realized by provider ($FSre = FS\min$), meaning sufficient *actual* effectiveness of functionality requirements execution, that is:

$$CDFSre(\min) = (FS\min / FS\max) \times 100\%,$$

- $CDFSre(\max)$ – maximum accepted compatibility degree of functionality realized by provider ($FSre = FS\max$), that is:

$$CDFSre(\max) = (FS\max / FS\max) \times 100\% = 100\%.$$

2. *CDEre* is within the allowed range, that is:

$$CDEre(\min) \le CDEre \le CDEre(\max),$$

where:

- $CDEre(\min)$ – minimum accepted compatibility degree of work effort realized by provider ($Ere = E\max$), meaning sufficient *actual* effectiveness of work effort requirements execution, that is:

$$CDEre(\min) = (E\min / E\max) \times 100\%,$$

- $CDEre(\max)$ – maximum accepted compatibility degree of work effort realized by provider ($Ere = E\min$), that is:

$$CDEre(\max) = (E\min / E\min) \times 100\% = 100\%.$$

3. *CDPre* is within the allowed range, that is:

$$CDPre(\min) \le CDPre \le CDPre(\max),$$

where:

- $CDPre(\min)$ – minimum accepted compatibility degree of functional productivity realized by provider ($Pre = P\min$), meaning sufficient *actual* effectiveness of productivity requirements execution, that is:

$$CDPre(\min) = (P\min / P\max) \times 100\%,$$

- $CDPre(\max)$ – maximum accepted compatibility degree of functional productivity realized by provider ($Pre = P\max$), that is:

$$CDPre(\max) = (P\max / P\max) \times 100\% = 100\%.$$

These conditions allow to verify accuracy of prognoses and the execution level of provider's commitments to a client. From client's point of view, they enable to make rational *ex post* pricing of project product based on measurement of the realized functional size (*FSre*) and work cost per functionality unit formally agreed at the stage of provider selection (for more details see [21]). Thus client will pay for the actually delivered product functional size – and not for the functionality that was offered yet not realized by the provider.

Thus, functional assessment of BSS D&EP is conducive to making investment decisions that meet the criterion of effectiveness in the area of functional attributes, that is those contributing to delivering product compatible with client's rational requirements with regard to these attributes. This is possible thanks to:

- The possibility of determining sufficient expected effectiveness of functional attributes execution (indicators: $CDFSo(\min)$, $CDEo(\min)$, $CDPo(\min)$).
- The possibility to eliminate providers' offers that do not meet conditions of the sufficient expected effectiveness of functional attributes execution on the basis of the offered effectiveness indicators (*CDFSo*, *CDEo* and *CDPo*).
- The possibility to choose offer being best suited to client's rational requirements with regard to functional attributes, that is having the highest allowed offered effectiveness of required productivity execution (having the highest *CDPo*).

Thanks to the realized effectiveness indicators (*CDFSre*, *CDEre*, *CDPre*) and to comparing them against indicators of sufficient execution effectiveness ($CDFSre(\min)$, $CDEre(\min)$, $CDPre(\min)$), functional assessment allows also for the evaluation of the realized project effectiveness in terms of functional attributes.

4 Using Functional Assessment to the BSS D&EP economic efficiency evaluation

Assuming that the major benefit of the BSS D&EP execution to a client is functionality brought by the project, which is measurable with the use of product functional size for every BSS, then, without excessive simplification, it may be presumed that:

1. Fundamental effect arising from the BSS D&EP execution is the value of product functional size, that is the value of functional benefits brought about by project ($Efec_f$), being the product of this size (*FS*) and the value delivering by functionality unit ($V(fu)$).
2. Fundamental costs required for the BSS D&EP execution are total work costs of the project execution (*WC*), being the product of effort (*E*) and unit work cost calculated with regard to effort unit ($C(eu)$).

Therefore, in accordance with the formula (1), we will receive the following:

$$Ef = Efec_f / WC = (FS \times V(fu))/(E \times C(eu)).$$

On the whole, if a project was economically efficient the value of benefits coming from its execution would have to be higher than its costs and therefore general condition of BSS D&EP economic efficiency with the adopted assumptions will read as follows:

$$(FS \times V(fu))/(E \times C(eu)) > 1,$$

which means that:

$$P > C(eu)/V(fu).$$

Hence if functional productivity (*P*) is higher than the ratio of unit work cost calculated with regard to effort unit ($C(eu)$) to the value delivering by functionality unit ($V(fu)$) than, following the adopted assumptions, project having such productivity will be considered economically efficient. Value delivering by functionality unit must be therefore higher than the quotient of this unit work cost and productivity.

In the situation where in the market work costs for projects of similar characteristics are evened out, i.e., where unit work cost calculated with regard to effort unit ($C(eu)$) does not constitute factor differentiating offers of potential providers, minimum value delivering by functionality unit ($V(fu)$) necessary to ensure project efficiency depends on the productivity (P): the higher the productivity, the lower this minimum value. In this sense higher productivity allows for working out lower value by functionality unit to ensure project efficiency. When product functional size (FS) is known, determining minimum $V(fu)$ allows to determine minimum value of functional benefits necessary to ensure project efficiency.

5 Relationships between Functional Assessment and the BSS D&EP effectiveness and economic efficiency evaluation

Based on the adopted assumptions the following conclusions may be drawn:

1. Project execution variant consistent with functional assessment criteria will be also economically efficient if the value delivering by functionality unit ($V(fu)$) will be higher than work cost per functionality unit ($C(fu)$) offered in this variant, being determined by the offered work effort per functionality unit ($E(fu)$). It means that variant consistent with functional assessment criteria may not be economically efficient since there is no possibility for it to guarantee that functionality unit will produce value being sufficient to it – as this value depends on usage the realized functionality by client.
2. Economically efficient variant of project execution also will be consistent with functional assessment criteria if work cost per functionality unit ($C(fu)$) offered in this variant will be within the allowed range of values, resulting from the allowed range for functional productivity (see condition 3 in section 2), as well as it will meet adequate conditions concerning the offered product functional size (see condition 1 in section 2) and offered work effort (see condition 2 in section 2). It means that variant not consistent with FA criteria may in fact be economically efficient.
3. Among project execution variants consistent with FA criteria what proves being optimum variant with regard to this assessment is the one characterised by the highest offered functional productivity being within the allowed range, i.e., with the highest $CDPo$.
4. What proves being optimum variant in terms of economic efficiency in the case of efficient variants is variant having the lowest value delivering by functionality unit ($V(fu)$) necessary to ensure project efficiency, so the variant having the highest offered functional productivity (P), which means the lowest offered work effort per functionality unit ($E(fu)$).
5. Project execution variant being optimum with regard to functional assessment and economically efficient will also be optimum in terms of economic efficiency if the highest functional productivity offered in economically efficient variants is not higher than maximum allowed functional productivity (Pmax). Otherwise such variant will not be optimum in terms of economic efficiency – as the optimum variant will be the one characterised by the highest offered functional productivity, exceeding maximum allowed functional productivity.
6. Project execution variant being optimum with regard to economic efficiency and meeting criteria of FA will also be optimum in terms of functional assessment.

As indicated by the analysis, variant being optimum in terms of economic efficiency does not have to be optimum in terms of FA – and vice versa, but also economically efficient variant does not have to be the variant consistent with FA criteria, either – and vice versa. While functional assessment imposes additional limitations on the execution variants, having fundamental character from client's point of view as they concern the effectiveness of meeting his requirements with regard to FA attributes.

Hence taking into account only the criterion of economic efficiency, which with the given assumptions is boiled down to the evaluation of the offered functional productivity without considering the allowed range of values, client faces the risk of the lack of effectiveness in project execution. As the same productivity may be offered by execution variants that differ with regard to the offered functional size and offered work effort, being the attributes that may not correspond with client's rational requirements. If project product was to meet the required functions, it is necessary to deliver adequate functional size, being within the allowed tolerance interval, which on the other hand requires paying corresponding work effort, also with some tolerance.

Moreover, the author is of opinion that the chance for the execution of economically efficient project without ensuring its compatibility with the required product functionality, being the source of the expected benefits, as well as with the expected work costs, goes down. Hence one may formulate hypothesis that this is project effectiveness that decides on its efficiency. In addition, not taking into account the allowed ranges of values for the offered functional productivity increases the risk of overstating or understating the effort per functionality unit, which as a result may cause some negative consequences. From the viewpoint of economic efficiency, the most important among such consequences would be failure to deliver required functional size, caused by the overstated productivity, that is by the understated unit work effort ($E(fu)$), which in the situation where the unit work cost is independent on the product size will result in the necessity to increase the value delivering by functionality unit ($V(fu)$) necessary to ensure project efficiency. Therefore, despite the fact that work effort per functionality unit was understated by a provider, this is client who pays the consequences of such understating.

In addition, evaluation of economic efficiency in no way contributes to the reduction of negative phenomena, being common to the Polish BSS D&EP practice, which consist in: deliberate lowering of planned BSS execution costs by offerors in order to win the contract, uncontrolled increase in client's functional requirements as to the product not being correspondingly reflected in the costs of project as well as in non-rational *ex ante* and *ex post* product pricing. Since these phenomena promote exceeding of project execution costs and delivering of functionality lower than the required one, the chance for achieving expected economic efficiency drops. The FA approach verification proved that using this concept

allows limiting these negative phenomena (for more details see [20] and [21]).

Functional assessment, on the other hand, does not serve as a substitute for the evaluation of project economic efficiency as it does not allow to state unconditionally whether given execution variant is economically efficient, however it supports efficiency analysis. And this is because it allows for *ex ante* and *ex post* determining of project execution work costs, comparing them against the offered costs as well as determining – with the adopted assumptions at all its stages – of the project efficiency condition, i.e., the lowest value, which should be delivered by functionality unit to ensure project economic efficiency. This allows for determining minimum value of functional benefits being necessary to achieve this efficiency for the required product functional size. These activities enable to:

- Compare thus calculated value of benefits with potential value assumed by client
- Make it easier to assess the possibility to achieve value of functional benefits being necessary to ensure economic efficiency
- Compare execution variants from the point of view of expected and offered work costs and economic efficiency condition
- Select execution variant having the highest potential economic efficiency
- Compare expected value of functional benefits and work cost of the chosen execution variant against the value of actual benefits and costs.

Therefore estimation of BSS D&EP *economic efficiency should be supplemented with functional assessment, comprising also the evaluation of project effectiveness with regard to functional attributes and additionally, promoting the reduction of negative phenomena mentioned above*. Thus the two measurable criteria of the rational investment decision made with regard to the BSS D&EP will be fulfilled, which as a result should satisfy both the management of an investor and the users of future product.

6 Concluding remarks

Summing up it should be stated that the concept of BSS D&EP functional assessment proposed by the author, based on the FSM concept and methods, allows for:

- *Ex ante* and *ex post* evaluation of BSS D&EP effectiveness – thanks to the possibility of estimating compatibility degree of FA attributes offered by provider and possibility of measuring compatibility degree of FA attributes realized by provider with FA attributes required by client.
- Supporting evaluation of BSS D&EP economic efficiency – thanks to the possibility of *ex ante* and *ex post* determining of project work costs, comparing them against the costs offered by provider as well as determining of the project economic efficiency condition.

This is possible thanks to the FA capabilities, being proved in the verification carried out in practice on the basis of case study (see [20] and [21]), which revealed that FA allows, among others, for:

- Identification of functional benefits coming from the BSS D&EP execution and expressing these benefits quantitatively in the form of required, offered and realized product functional size.
- Identification of BSS D&EP work costs and expressing them quantitatively in the form of required, offered and realized project work effort.
- Justification of investment costs for BSS D&EP that need to be paid by client.
- Determining client's FA attributes requirements towards BSS D&EP provider in a way that is not only measurable, but also rational, which means that the expected FA attributes will be neither unrealistically overstated (product functional size, productivity) nor understated (work effort).
- Making rational investment decision by a client on engaging in the execution of BSS D&EP on the basis of initially estimated FA attributes.
- Making rational decision by potential providers on the offered work cost per functionality unit, that is on offering it on the level that is neither understated (the risk of exceeding this cost is passed to the provider) nor overstated (inflated level reduces the chance of choosing given provider's offer), what needs awareness of own productivity and proper productivity management, thus needs improvement of software processes.
- Client's evaluation of the submitted offers for BSS D&EP execution and selection of offer (offers) being most suited to his requirements in terms of FA attributes.
- Comparing variants of BSS D&EP execution from the viewpoint of the estimated work costs and economic efficiency condition as well as selecting variant having the highest potential efficiency.
- Formal *ex ante* pricing of BSS based on the estimated required functional size and on the work cost per functionality unit offered by the chosen provider and approved by client, therefore mutually agreed.
- *Ex post* pricing of the delivered BSS on the basis of actually realized product functional size and work cost per functionality unit, having been mutually and formally agreed by client and provider at the stage of choosing provider.
- Evaluation of the realized BSS D&EP with regard to the compatibility with client requirements concerning FA attributes and verification of accuracy of prognoses concerning FA attributes along with determining of the potential causes of discrepancies between the actual and estimated values.
- Collecting own benchmarking data, expressing dependencies being specific to the given provider, what allows to reduce the risk of insufficiently accurate estimation for the FA attributes of BSS D&EP that are going to be realized in future.
- Improving of FSM methods and effort estimation models and thus software development and enhancement processes.

In the case of projects for which it is possible to make objective and reliable economic efficiency estimation, functional assessment, concerning first of all effectiveness of the functional requirements execution, should be treated as supplementary – thus the two measurable criteria of rational investment decision will be fulfilled. This is because the efficiency evaluation does not allow for

assessing the effectiveness of functional requirements execution, neither it contributes to reducing the mentioned above negative phenomena occurring in the practice of BSS D&EP execution. What is more, in view of these negative phenomena, actual project efficiency probably will be below the estimated one.

Functional assessment, on the other hand, is not a substitute for the evaluation of economic efficiency although it does support its analysis. This is because it allows for *ex ante* and *ex post* determining of project work costs and, having adequate assumption adopted, it also enables to answer the question about the condition, which should be fulfilled so that a specified execution variant is economically efficient. As a result it enables to determine values of functional benefits being necessary to ensure economic efficiency of the required functionality execution. Thus it makes it easier to evaluate the chance of achieving such benefits as well as it enables to assess particular execution variants in this respect and indicate variant characterized by the highest potential economic efficiency. What is more, project product's compatibility with required functionality increases the chance to achieve assumed benefits whereas project's compatibility with the expected work effort with taking into account assumed benefits – to execute project that is economically efficient. However, it does not allow to state categorically *whether* or not given execution variant is economically efficient but only to specify *when* it is going to be economically efficient. Thus the usefulness of FA comes into view particularly in the case of these projects for which objective and reliable *ex ante* proving of their economic efficiency is very hard or controversial, or it is not of key importance (e.g., some projects which are executed in public administration) or not justified (e.g., the so-called obligatory projects, being undertaken to make it possible for a company to survive on the market).

Conclusions coming from the above analysis clearly indicate, that BSS D&EP FA concept can constitute the basis for rational decisions not only for BSS providers, but also for BSS clients. Thus this paper, by linking the FSM issues with economic aspects through the FA concept, may contribute to better understanding of the FSM importance, still being underestimated by business managers, as in the subject literature these issues are usually considered from the technical perspective.

The surveys on these issues will be continued while the research area will be extended to other economic BSS D&EP aspects.

7 References

[1] M. Moszoro. „Partnerstwo publiczno-prywatne w monopolach naturalnych w sferze uzytecznosci publicznej" [„Public-private Partnership in Natural Monopolies in the Sphere of Public Utility"], Warsaw School of Economics Publishing House, Warsaw, 2005.

[2] A. Brown. "IS Evaluation in Practice"; in: The Electronic Journal Information Systems Evaluation, vol. 8, no. 3, pp. 169–178, 2005.

[3] E. Frisk, A. Plantén. "IT Investment Evaluation – A Survey of Perceptions Among Managers in Sweden", in: Proceedings of the 11th European Conference on Information Technology Evaluation, pp. 145–154, Academic Conferences, 2004.

[4] Z. Irani, P. Love. "Information systems evaluation: past, present and future", in: European Journal of Information Systems, vol. 10, no. 4, pp. 183–188, 2001.

[5] S. Jones, J. Hughes. "Understanding IS evaluation as a complex social process: a case study of a UK local authority", in: European Journal of Information Systems, vol. 10, no. 4, pp. 189–203, 2001.

[6] A.J. Silvius. "Does ROI matter? Insights into the True Business Value of IT"; in: The Electronic Journal Information Systems Evaluation, vol. 9, issue 2, pp. 93–104, 2006.

[7] B. Czarnacka-Chrobot. "Analysis of the Functional Size Measurement Methods Usage by Polish Business Software Systems Providers"; in: Software Process and Product Measurement, A. Abran, R. Braungarten, R. Dumke, J. Cuadrado-Gallego, J. Brunekreef, Eds., Lecture Notes in Computer Science, vol. 5891, pp. 17–34, Springer-Verlag, Berlin-Heidelberg, 2009.

[8] Standish Group. "CHAOS Summary 2009", West Yarmouth, Massachusetts, 2009.

[9] Standish Group. "Modernization – Clearing a pathway to success", West Yarmouth Massachusetts, 2010.

[10] T.C. Jones. "Patterns of software systems failure and success", International Thompson Computer, Boston, 1995.

[11] Panorama Consulting Group. "2008 ERP Report, Topline Results", Denver, 2008.

[12] ISO/IEC 14143 Information Technology – Software measurement – Functional size measurement – Part 1-6, ISO, Geneva, 1998-2007.

[13] ISO/IEC 19761:2011 Software engineering – COSMIC: a functional size measurement method, ISO, Geneva, 2011.

[14] ISO/IEC 20926: 2009 Software and systems engineering – Software measurement – IFPUG functional size measurement method 2009, ISO, Geneva, 2009.

[15] ISO/IEC 20968:2002 Software engineering – Mk II Function Point Analysis - Counting practices manual, ISO, Geneva, 2002.

[16] ISO/IEC 24570:2005 Software engineering – NESMA functional size measurement method version 2.1 - Definitions and counting guidelines for the application of Function Point Analysis, ISO, Geneva, 2005.

[17] ISO/IEC 29881:2008 Information Technology – Software and systems engineering – FiSMA 1.1 functional size measurement method, ISO, Geneva, 2008.

[18] State Government of Victoria. "southernSCOPE, Reference Manual", Version 1, Government of Victoria, Melbourne, Australia, 2000.

[19] Finnish Software Metrics Association. "nothernSCOPE – customer-driven scope control for ICT projects", FiSMA, Finland, 2007.

[20] B. Czarnacka-Chrobot. "Methodologies Supporting the Management of Business Software Systems Development and Enhancement Projects Functional Scope", in: Proceedings of the 9th International Conference on Software Engineering Research and Practice (SERP'10), The 2010 World Congress in Computer Science, Computer Engineering & Applied Computing (WORLDCOMP'10), H.R. Arabnia, H. Reza, L. Deligiannidis, Eds., pp. 566-572, CSREA Press, Las Vegas, Nevada, USA, 2010.

[21] B. Czarnacka-Chrobot. "Rational Pricing of Business Software Systems on the basis of Functional Size Measurement: a Case Study from Poland", in: Proceedings of the 7th Software Measurement European Forum (SMEF 2010), T. Dekkers, Ed., pp. 43–58, Libreria Clup, Rome, Italy, 2010.

Estimating and decision making for Design projects and Cognitive Bias of Hyperbolic Discounting

Neha Srivastava,

Lancaster University Management School, Lancaster, LA2 0PF U.K

Email: neha0486@gmail.com

Keywords: Hyperbolic Discounting, Cognitive Bias, Decision Making, SCI model, Software Design

The 2011 International Conference on Software Engineering Research and Practice (SERP'11)

1. Introduction

Human beings are conditioned and designed by society to relate high degree of uncertainty to future. Several behavioral studies have demonstrated that the phenomenon of attaching hazard rate to future events is consistent across habits related to food, drugs, exercising, impulsive shopping and saving money. A person, when planning for long term would think about health and monetary benefits arising out of all of these. But when faced with a choice would prefer to procrastinate. There is sufficient evidence for the fact that humans and animals tend to be more impatient when dealing with reward nearer in time than one due further in future. This phenomenon is called as Hyperbolic Discounting. This paper explores the hyperbolic discounting as a cognitive bias.

Hyperbolic discounting can be defined as the act of valuing intertemporal choices on a hyperbolic scale. Renowned economist Robert Strotz (1956) conducted empirical studies on agents whose preferences change over time. He found that when faced with a choice between two rewards of different values due at different time, an individual tends to choose one which is due after a shorter time delay even though its value is comparatively lesser than the other one which is due at a later stage. As the time when the rewards are due increases person tends to be more patient in choosing the reward. Loewenstein and Thaler (1989) classify such behavior as 'extremely myopic preferences'.

Angeletos, et al. (2001) present an interesting example of a worker "who prefers a 20-minute break in 101 days to a 15-minute break in 100 days. But when both rewards are brought forward in time, preferences exhibit a reversal, reflecting more impatience; the same person would prefer a 15-minute break right now to a 20-minute break tomorrow".

This paper explores the concept of hyperbolic discounting as a bias which leads to irrational behavior from human agents while designing projects. An integrated approach towards project management and human need for instant gratification as a behavioral pattern can prove to be very resourceful for long-term planning and determining strategies, estimates related to deadlines and revenues for critical projects. This paper explores the design approach which would result in minimum deviation from the planned behavior of the model if such intertemporal choices and '*myopic preferences*' (Loewenstein and Thaler, 1989) are taken into consideration.

2. Literature Review

Several researches have been conducted to understand the disconcertingly alluring idea of instant gratification and discounting time at a hyperbolic rate. Robert H. Strotz(1956) was amongst the first few to conduct empirical studies on *dynamically inconsistent* choices due to hyperbolic discounting. He proposed use of commitment as an instrument against tendency to discount time at hyperbolic rate, such as mandatory social security requirement or financial instruments. Victorian economist Jevons (1871/1911 cited in Ainslie, Haslam, 1992: 60) suggests that " To secure a maximum of benefit in life , all future events, all future pleasures or pains,

should act upon us with same force as if they were present, allowance being made for their uncertainty . The factor expressing the effect of remoteness should, in short, always be unity, so that time should have no influence. But no human mind is constituted in this perfect way, a future feeling is always less influential than a present one". This brings us to the concept of debiaising and rebiasing (Larrick, 2004). It refers to input of efforts to reduce the illusion of delayed incentives. Debiasing requires effort or corrective action from the agent at present to change the future value of the object. Rebiasing simply involves a "no-effort" strategy and is implemented when people get tired of putting efforts. This theory helps us to formulate the design approach in which will effort is required at time when decisions have to be made.

Cropper and Laibson (1999) present a good argument on the use of hyperbolic discount functions by managers and decision-makers instead of constant exponential discounting techniques. Their research also suggests the intervention by Government in order to protect consumers from making impulsive choices to satisfy their urge for instant gratification. This has direct implication on project managers who choose constant time discounting instead of hyperbolic discounting techniques for cost-benefit analysis.

Laibson cites Strotz(1956) on the commitment device to improvise or rectify the *self defeating behavior.* This research was a breakthrough in itself to bring the change from constant discounting to hyperbolic discounting, but it simply accepts this bias as something which cannot be isolated from human nature and gives no solutions for the managers.

Newell and Pizer (2003, pp. 52-71) state that "Implicit in any long-term cost-benefit analysis is the idea that costs and benefits can be compared across long periods of time using appropriate discount rates". Based on the research by Arrow and Kurz(1970, pp. 331-334) the managers follow the concept of time discounting using a constant exponential discount rate. But empirical research shows that people tend to discount the future hyperbolically, by using different discount rates inversely proportional to time. Lesser the time (when return would be attained) higher the discount rate and vice versa (Ainslie, 1992; Cropper, Portney and Aydede, 1994). Thaler contrasts constant and hyperbolic discount rates and considers that using a constant discount rate, "a reward which loses 10% of its value in 1 year, will lose an additional 10% of its value if delay is increased to 2 years but with hyperbolic discounting, the proportional decay in reward value changes with each new time period and a reward loses more of its value during initial delays than it does in later delays, this phenomenon is referred to as *dynamic inconsistency".* The next section discusses the implications of hyperbolic discounting on Design projects and explores how to minimize the effect of bias using Design models.

3. Estimates and Design with Hyperbolic Discounting as a Cognitive Bias

Projects are evolutionary and complex systems. Researchers indicate (Hart, 1982; Shooman, 1983; Brooks, 1978) that actions and decisions taken by people in project situations are significantly influenced by pressures and perceptions produced by project's schedule. Laibson(1997) proposed a mathematical model for hyperbolic discounting. According to his model exponential discount function, δ^{τ} is known to have a constant discount rate , $\log(1/\delta)$, where τ is the instantaneous discount rate. However, according to Laibson(1997, pp 450) " generalized hyperbolic discount function is characterized by an instantaneous discount rate that falls as τ rises using equation $\gamma / (1 + \alpha \tau)$, with $\alpha, \gamma > 0$". Figure 1 shows the exponential discount function (assuming that $\delta = 0.97$) alongside the generalized hyperbolic discount function (assuming that $\alpha = 10^4$, and $\gamma = 5*10^3$).

According to Devenny(1976, cited in Abdel- Hamid , Madnick, 1984, p.15), past experiences indicate a strong bias on the part of software developers to underestimate the scope of a software project. Project managers make use of 'Safety Factor' for estimates to reduce the effect of bias (Abdel-Hamid, Madnick, 1985). We can try to minimize this 'Safety factor' or level of contingency by use of a design model which has an *Action-centric approach* and not a *Reason-centric approach*. The rationale behind such a proposal is the *dynamically inconsistent* human behavior which can be attributed to cognitive bias of discounting time hyperbolically.

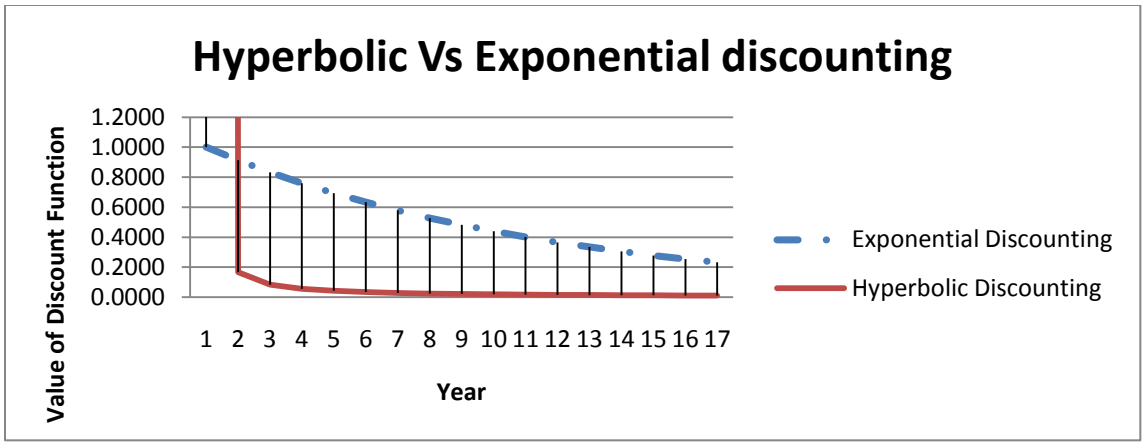


Figure 1. Hyperbolic discounting and Exponential Discounting Source : Laibson (1997)

Ralph (2010, pp.2) states that 'Technical Rationality and the Technical Problem-solving paradigm are consistent with the *cognitivist* view of human action, wherein actions are executed and understood through a plan..... Plans are prerequisites to action. Unanticipated conditions trigger replanning; evaluation is performed by comparing resulting and planned actions and outcomes'. He proposed an *action-centric* model for software design processes which is called Sensemaking- Coevolution-Implementation. Figure 2. is an overview of the model , the components of which are described in Table 1 (in the Appendix A). This model gives the designer the flexibility to alternate between framing, making moves and evaluating the moves.

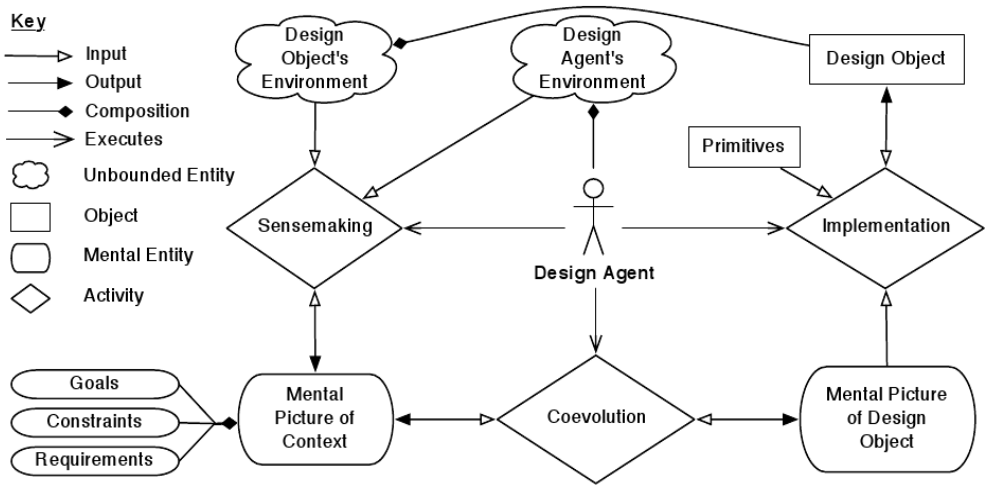


Figure 2: The Sensemaking-Coevolution-Implementation Framework , Source (Ralph,2010b)

Managers tend to follow *naturalist way* of decision-making, the popular term for this is 'gut feeling'. The decision-making paradigm for SCI model is *naturalistic* and not *rational*. Hyperbolic discounting will result in skewed estimates as the need for instant gratification clouts the cognition due to which designer would not be able to view the bigger picture of the problem at hand. Also, while analyzing risks people tend to be more concerned with the risks pertinent to present situation than one which might be faced later. Other frameworks for software design have little or no scope of revaluating the decisions .These models work under illusionary belief that all the decisions are *rational*. The iterative nature of Sensemaking-Coevolution-Implementation framework gives the designer chance to make amendments to the design and the behavioral properties of model.

The *action-centric* approach of SCI model with *reflection-in-action* as core paradigm is consistent with the game played by quasi-hyperbolic agents proposed by Phelps and Pollak(1968), Strotz(1956), Laibson(1998). Laibson's (1998) model supposes an individual as a composite of autonomous temporal selves. These selves then interact as players in a finite-horizon dynamic game. For consideration in our software design approach using SCI framework we can use Laibson's approach of using autonomous temporal selves for Subgame Perfect Equilibrium (SPE) .These agents interact to reach an equilibrium state, a decision which is intersection of the decision of all the players (Temporal selves) . This decision is represented by S = $\prod_{t=1}^{t=n} S_t$. (Figure 3 has been drawn to represent the intersection as perfect decision). The situation under which all the human game players (H) would be honest (Laibson, 1975, pp. 451) about their choices, would approach a near perfect decision or equilibrium.

Figure 3 : Subgame Perfect Equilibrium represented by S = $\prod_{t=1}^{t=n} S_t$

4. Recommendations

The paper proposes modification of SCI framework Ralph(2010) by inclusion of game playing temporal selves as decision agents . The model suggests that Sensemaking and Decision making should not be intertwined (refer Figure 4). Decision making should be performed iteratively every time after Sensemaking is performed. Next step is drawing a mental picture of the context. There is clearly a need to evaluate the decision on which work will be performed in the implementation step before co-evolution. Since with hyperbolic discounting as a bias the preference might change in the process of implementation without being noticed, so clearly identifying the decision is very essential.

To isolate the decision from cognitive bias, decision making should be performed after the Sensemaking is complete, so that design object and design agent environment does not directly influence the decision. The decision makers should work on the mental picture of the context. Sub-game perfect equilibrium will be attained by working on the intersection of decisions taken at the following steps: when goals of the system are defined, when implementation begins, when tests are performed, when deployment takes place. The preference change can be monitored more closely by Sensemaking to make adjustments. This model would minimize the difference in expected behavior and real behavior of design projects and is based on debiasing. There is much scope for further research since this is a theoretical model with no empirical research to support it. Also, the question that whether Hyperbolic discounting is altogether an undesired phenomenon and always results in negative outcomes due to biased decisions is open for debate. The design model is not generic, and its application to projects other than software projects can be explored.

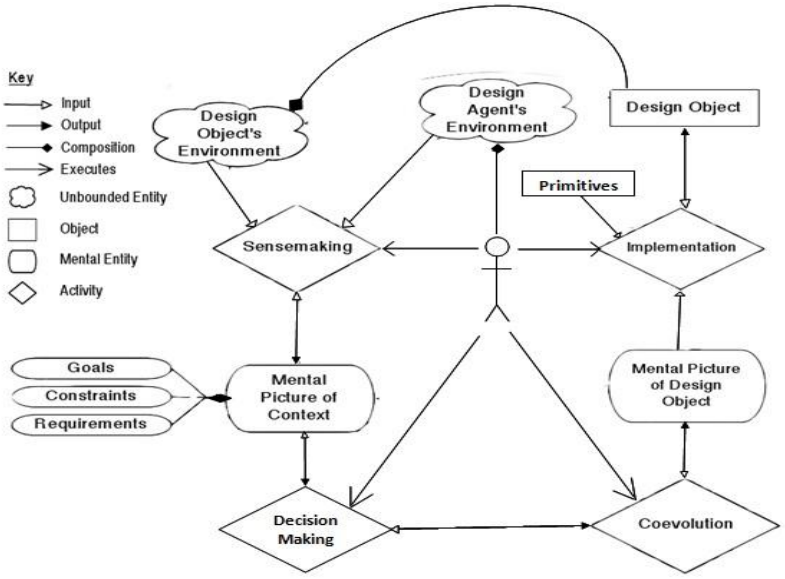


Figure 4: Modification in SCI model to include Decision Making as one of the activity

5. Conclusions

Dynamic inconsistency of human behavior which arises due to cognitive bias of hyperbolic discounting can be detrimental for decision making process. The existing software design models tend to consider the human behavior as rational which in not true in real world. The model for design should allow the agents to behave, decide and function in their natural way. The SCI model is action centric which is more suitable for naturalistic design agents, and hence is a good choice for design project. The paper proposes slight change in SCI model and proposes inclusion of Decision Making as a step. Use of Sensemaking-Decisionmaking-Coevolution-Implementation model allows the decision makers to iterate and re-evaluate the decision.

Appendix A

Table 1: Concept of Sensemaking-Coevolution-Implementation, Source: (Ralph,2010)

Concept	Meanings
Constraints	A restriction on a structural or behavioral property of *design* object
Design Agents	An entity or group of entities that is capable of forming intentions and goals and taking actions to achieve those goals, and that specifies the

	structural properties of the design object
Design Object's Environment	The totality of surroundings in which the design object exists or is intended to exist.
Design Agent's Environment	The totality of surroundings of the design agent
Design Object	A (possibly incomplete) manifestation of the mental picture of design object, composed of primitives, in the design object's environment
Goals	Optative statements (which may exist at varying level of abstraction) about the effects the design object should have on the design object's environment.
Mental Picture of Context	The collection of all beliefs, held by the design agent, regarding the design agent's environment and the design object's environment.
Mental Picture of Design Object	The collection of all belief held and decision made by the design agent concerning the design object.
Primitives	The set of entities from which the design object may be composed
Requirements	A structural or behavioral property that a design object must possess
Sensemaking	The process where the design agent perceives its environment and the design object's environment and organizes these perceptions to create or refine the mental picture of context.
Coevolution	The process where the design agent simultaneously refines its mental picture of design object based on its metal picture of context, and vice versa
Implementation	The process where the design agent generates or updates a design object using its mental picture of design object.

References

Abdel-Hamid,T.K and Madnick, S.E. (1986). Impact of schedule estimation on software behavior . *IEEE Software*. 3 (4), p70-75.

Ainslie, G. (1974), Impulse control in Pigeons, Journal of the experimental analysis of behavior, Harvard University, 21, p485-489.

Ainslie, G. (1991), Intertemporal Choice: Derivation of "Rational" Economic Behavior from Hyperbolic Discounting, The American Economic Review, Papers and Proceedings of the Hundred and Third Annual Meeting of the American Economic Association. 81(2), p334-340.

Ainslie, G. (1992). Hyperbolic Discounting. In: Loewenstein, G. and Elster, J. Choice over time. New York: Russel Sage Foundation. p57-92.

Afzar, O. (1999) Rationalizing hyperbolic discounting, Journal of Economic Behavior & Organization. 38(2), p245-252.

Angeletos, G. , Laibson, D. , Repetto A. , Tobacman, J. and Weinberg, S. (2001) , The Hyperbolic Consumption Model, J. Econ. Perspect. p15-47.

Cropper, M.L. and Laibson, D. (1999). In: Portney P.R. , Weyant J.P. Discounting and Intergenerational Equity. Washington: Resources for the Future.

Harris, C. , Laibson, D. (2001), Dynamic choices of hyperbolic consumers, Econometrica 69 (5) , p935-957.

Hantula, D. A., and Bryant, K. (2005). Delay discounting determines delivery fees in an e-commerce simulation: A behavioral economic perspective. Psychology & Marketing, 22, p153-162.

Hart, J.J. (1982), The Effectiveness of Design and Code Walkthroughs, The 6th International Computer Software and Applications Conference, COMPSAC.

Krutchen, P.(2005), Casting Software Design in the Functio-Behavior-Structure Framework. IEEE Software, 22, p52-58.

Laibson, David I. (1997), Golden Eggs and Hyperbolic Discounting, Quarterly Journal of Economics. 62(2), p443-478.

Larrick, Richard P. (2004), "Debiasing," in *Handbook of Experimental Psychology*, Ed. Derek Koehler and Nigel Harvey, Blackwell Publishing.

Loewenstein, G. and Prelec, D. (1992), Anomalies in Intertemporal Choice: Evidence and an Interpretation, Quarterly Journal of Economics, 57, p573-598

Lowenstein, G. and Thaler, Richard H. (1989), Anomalies: Intertemporal Choice, Journal of Economic Perspectives, 3, p181-193.

Newell, R.G, and Pizer, W. (2003). Discounting the distant future: How much do uncertain rates increase valuations? Journal of Environmental Economics and Management. 46 (1), p52-71.

Phelps, E.S., and Pollak, R.A. (1968) , On Second-best National Saving and Game-equilibrium Growth, Review of Economic Studies, 35, p185-199.

Ralph, P. (2010a), Comparing two software design process theories, In: Proceedings of the International Conference on Design Science Research in Information Systems and Technology (DESRIST 2010), LNCS 6105, Springer, St. Gallen, Switzerland, p. 139-153.

Ralph, P. (2010b), The sensemaking-coevolution-implementation framework of software design, MIS Quarterly((under review)), p76pages.

Strotz, Robert H. (1956), Myopia and Inconsistency in Dynamic Utility Maximization, Review of Economic Studies, 23, p165-180.

Thaler, Richard H. (1981), Some Empirical Evidence on Dynamic Inconsistency, Economic Letters, 8, p201-207.

Thaler, Richard H. and H.M. Shefrin. (1981). An Economic Theory of Self Control, Journal of Political Economy, 89, p392-410.

Lowenstein, G . and Thaler, R . H. (1989), Anomalies: Intertemporal Choice, Journal of Economic Perspectives, 3, p181-193.

Usability Evaluation of Indian Academic Web Application Using LSP

Manju Pandey[1], **Priyanka Tripathi**[2], **S.C.Srivastava**[3]
[1,2]Department of Computer Application,
National Institute of Technology, Raipur, C.G, INDIA
e-mails: manjutiwa@gmail.com, priyanka_tripathi@hotmail.com
[3]Department of Electronics and Communication Engg.
MANIT, Bhopal, M.P, INDIA
e-mail: scs_manit@yahoo.com

Abstract – *This paper presents important factors affecting the Usability of Academic web-sites. Well known academic sites have been evaluated for their usability. Usability is the most important quality factor a web-site should consist of. A broad, integrated, engineering-based approach has been used to evaluate the usability of the Web-sites. A small, controlled and well planned experiment is conducted in order to evaluate the usability factors using Logic Scoring Preferences (LSP) grounded on continuous preference logic as mathematical background.*

Keywords: Academic, LSP, Quality, Usability, Website

1 Introduction

The purpose of this paper is to evaluate the Indian Academic websites QUIS method is used in order to collect the data. Students participated in the survey in order to evaluate the Usability of the academic site. Apart from it LSP method to perform the calculative and logical measurements. Usability is considered a fundamental factor of Web applications' quality because users' acceptability of Web applications seems to rely strictly on the applications' usability [1].ISO 9241 standard defines usability as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [2].

The first step in every evaluation process is the identification of all attributes that affect the ability of the evaluated system to satisfy requirements specified by a decision maker [3], [4],[5], [6],[8]. In the case of a car buyer, such attributes may include the power of engine, acceleration, the number of airbags, the number of doors, the volume of trunk, etc. After defining all relevant attributes, the decision maker specifies requirements for each attribute. For example, suppose that the volume of trunk that is less than or equal to a specific minimum value is not acceptable, and the volume that is greater than or equal to completely satisfies all user's requirements. The level of satisfaction can be analytically expressed as the following elementary (attribute) criterion function:

$$x=g(V) = \max\{0,\min[1,(V - Vmin)/(Vmax - Vmin)]\} \quad (1)$$

Therefore, for any volume of trunk, the level of satisfaction will belong to a unit interval. The value 1 denotes the complete satisfaction of user requirements and 0 denotes a complete failure to satisfy requirements. The values denote partial satisfaction of requirements. The variable is called the *preference score*, or simply the *preference*. It is interpreted as the degree of truth in the statement asserting that the evaluated system completely satisfies the requirement. Of course, it can also be interpreted as a grade of membership of the analyzed system in a fuzzy set of systems that completely satisfy the -requirement.

Many websites are being to accessed by the user as per their requirement. Websites such as bank, government organization, academic, online shopping and few more are the most which is being accessed. In order to evaluate the usability of the website different approach and methods are implemented. The most frequently used the usability measuring technique is Logic Scoring Preferences (LSP).

Educational Website users are mainly concerned with the following two major questions:

1. Can I find the information I am looking for in my website easily?
2. Can I find the information in timely manner? [7]

Here we have taken two Indian academic websites. These website are one of the most visited by the students. Being the educational domain website certain criteria has been delivered to the user.

2. Related work

Usability characteristic is mainly divided into Inspection *Methods* (without end user) and *test user* (with end user) [9]

Usability Inspection Method Characteristic

This is a set of methods for identifying usability problems and improving the usability of an interface design by checking it against established standards. These methods include heuristic evaluation, cognitive walkthroughs, and action analysis. Heuristic evaluation (HE) is the most common informal method. It involves having usability specialists judge whether each dialogue or other interactive element follows established usability principles [10].

Heuristic Evaluation Method: Heuristic Evaluation Method [11] is a usability engineering method for finding the usability problems in a user interface design so that they can be attended to be as part of an iterative design process. Heuristic evaluation involves having a small set of evaluators examine the interface and judge its compliance with recognized usability principles (the "heuristics"). Usually at least three usability experts evaluate the system with reference to established guidelines or principles, noting down their observations and often ranking them in order of severity. It is a quick and efficient method which can be completed in a few days.

Cognitive Walkthrough Method: The cognitive walkthrough is practical evaluation technique grounded in Lewis and Polson's [12] and it is a task-oriented method by which the analyst explores the system functionalities; that is, Cognitive Walkthrough simulates step-by-step user behavior for a given task. Normally one or two members of the design team will guide the walkthrough, while one or more users will comment as the walkthrough proceeds.

Action Analysis Method: The Action Analysis Method is divided into formal and back-of-the-envelope action analysis; in both, the emphasis is more on what the practitioners do than on what they *say* they do. The formal method requires close inspection of the action sequences a user performs to complete a task [8]. It involves breaking the task into individual actions such as move-mouse-to-menu or type-on-the-keyboard and calculating the times needed to perform the actions. Back-of-the-envelope analysis is less detailed and gives less precise results, but it can be performed much faster.

Table 1. Usability Evaluation Method

	Inspection Methods			Test Methods		
	Heuristic Evaluation	Cognitive Walkthrough	Action Analysis	Thinking Aloud	Field Observation	Questionnaires
Applicably in Phase	all	all	design	design	final testing	all
Required Time	low	medium	high	high	medium	low
Needed Users	none	none	none	3+	20+	30+
Required Evaluators	3+	3+	1-2	1	1+	1
Required Equipment	low	low	low	high	medium	low
Required Expertise	medium	high	high	medium	high	low
Intrusive	no	no	no	yes	yes	no

Usability Test User Characteristic

Thinking Aloud Method: is a very efficient way of getting a lot of qualitative data from a user [13]. As the name suggests, the user should think aloud while performing some specified task with the system. By verbalizing their thoughts, test users enable us to understand how they view the computer system. The users usually get a task to perform and are asked to think aloud while doing this. The experimenters often need to prompt the user to think out loud by asking questions like "What are you thinking about now?" and "What do you think this message means?" The experimenters are not supposed to answer any questions or draw the attention of the user to certain aspects of the interface that the user is not clearly working with.

Observation Method: is the simplest of all methods, it involves an investigator viewing users as they work and taking notes on the activity that takes place [11]. Observation may be either direct, where the investigator is actually present during the task, or indirect, where the task is viewed by some other means such as through use of a video recorder. Observation focuses on major usability catastrophes, which tend to be so glaring, are obvious the first time when they are observed and thus do not require repeated perusal of a recorded test session.

Questionnaire Method: is a method for the elicitation, and recording, and collecting of information [15]. The main of this advantage of this method is that a usability questionnaire gives you feedback from the user perception. If the questionnaire is reliable, and you have used it. According to the instructions, then this feedback is a trustworthy sample of what you will get from your whole user population. This method needs sufficient responses to be significant (we are of the opinion that 30 users is the lower limit for a study); and it identifies fewer problems than the other methods [11].

The MILE (Milano-Lugano evaluation method), by Paolini et al. [13], combines inspection methods with empirical tests.This method is strongly bonded to the usability concern, and almost completely dependent on human testers (domain analysts or final users).The MILE+ is an evolution of this method that presents a distinction between application dependent and independent analysis [14]. It considers two concepts: *abstract tasks* that can be applied to a number of applications and *concrete tasks* that are application-specific. The dimensions of this method are: *Contents, Services, Navigation, Interface* and *Graphic*. The MILE+ introduces technical evaluation, related with specific parts of the application, final user evaluation, and scenario based evaluation which is a domain-dependent evaluation. This method was used in the museums domain and, according to their authors, is partly automatable.

MiLE+ is the fruit of common research performed by TEC-Lab (University of Lugano) and HOC-Lab (Politecnico of Milan). MiLE+ tries to mix together some features of the

methods above presented stressing the strengths and minimizing the drawbacks.

In particular, MiLE+ is a usability inspection framework for web applications that strikes a healthy balance between heuristic evaluation and task-driven techniques. Clearly MiLE+ is not merely the sum of the more interesting characteristics of other methods, but it also introduces a new conceptual approach and several tools.

3. Evaluation criteria

We have proposed a quality tree with various factors and their sub factors along with the metrics for the overall quality for the academic site from or earlier research and through our experience [16]. Also we have used LSP method for evaluating quality y of Web-sites from user's point of view [17] [18].Details of LSP model has been given in section 3. We use the tree from our previous work. The evaluation process is done by using small, controlled and well planned experiments. The process used students of Master of Computer Applications of National Institute of Technology, Raipur, India. They were given a list of academic web-sites and were provided the evaluation scale for each metric. Table 3 gives a list of factors those were controlled during the experiment.

For each attribute A_i we have associated variable X_i which can take a real value by means of the elementary criteria function. The final result represents a mapping of the function value into elementary quality preference. The elementary quality preference is interpreted as a continuous logic variable. The value 0 denotes that X_i does not satisfy the requirements, and 1 denotes a perfect satisfaction of requirements. The values between 0 and 1 denote a partial satisfaction of requirements. Consequently, all preferences are frequently interpreted as a percentage of satisfied requirements, and defined in the range of [0,100%]. Students were provided these measurement criteria. The results are discussed in the next section.

4. Results and discussion

The Global Quality preference will be calculated using the Logic Score Preferences once these factors have been evaluated.

In this work, we have presented a systematic and quantitative engineering-based approach to evaluate and compare usability factor of Academic Web-sites. Based on this evaluation. criterion the overall ranking of the Indian Academic Web-sites according to their quality will be given. We are in the process of evaluating Global Quality Preference based on all four factors- usability, functionality, reliability and efficiency. The results will be published soon. Further, we can model simultaneity, replaceability, neutrality, and symmetric and asymmetric attribute relationships using logic aggregation operators.

Figure 1 Usability of the Selected Site

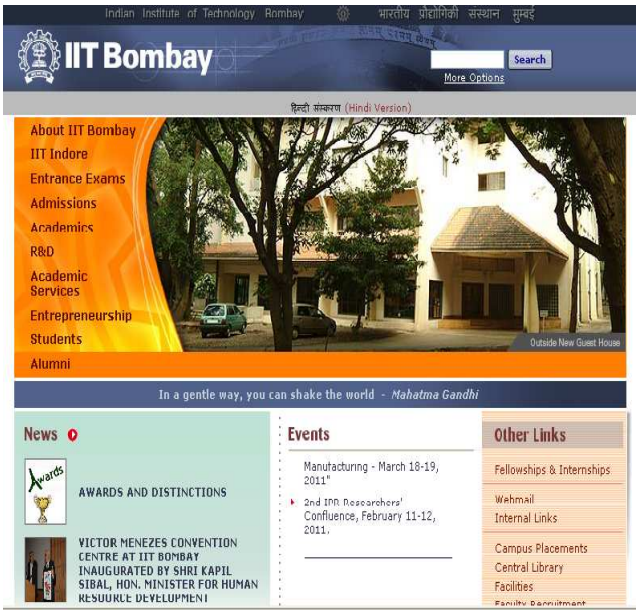


Figure 2 Usability of the Selected Site

Table 2. Usability of the selected web-sites

S. No	URL of selected Academic web-site	Usability (%)
1	http://www.iitb.ac.in	88
2	http://www.iitd.ac.in	78

Table 3. List of Factors in controlled Experiment

		IITB	IITD
1.1	Global site Understandability		
1.1.1	Site Map	1	1
1.1.2	Table of content	0.7	0.5
1.1.3	Link Related Features		
1.1.3.1	No of Link not working	0.3	0.5
1.1.3.2	Quality of Link Titles	0.7	0.6
1.1.4	Student Related Info	0.7	0.7
1.2	On- line feedback and Help Features		
1.2.1	Quality of help features		
1.2.1.1	Student oriented explanatory help	0	0
1.2.1.2	General Help	0.4	0.3
1.2.1.3	Search Help	0.3	0.2
1.2.2	Web-site Last Update Indicator		
1.2.2.1	Overall update date of the site	0	1
1.2.2.2	Scope wise update date of the notices	1	0
1.2.3	Addresses Directory		
1.2.3.1	E-mail Directory	0	0
1.2.3.2	Phone-Fax Directory	1	0
1.2.4	FAQ Features	0	1
1.2.5	On-line Feedback	0	1
1.3	Interface and Aesthetic Features		
1.3.1	Path Indicator	0	0
1.3.2	Presentation Permanence and Stability of Main controls	0.5	0.4
1.3.3	Style Issues		
1.3.3.1	Link Color Style Uniformity	0.6	0.5
1.3.3.2	Global Style Uniformity	0.7	0.5
1.3.4	Standard Aesthetic Features	0.6	0.5
1.4	Miscellaneous Features		
1.4.1	Foreign Language Support	0	0
1.4.2	New & Forthcoming	1	1

6. References

[1] Renato Otaiza, Cristian Rusu, Silvana Roncagliolo "Evaluating the Usability of Transactional Web Sites", 2010 Third International Conference on Advance in Computer-Human Interactions.

[2] M. Matera, F. Rizzo, G. T. Carughi, "Web Usability:Principles and Evaluation Methods" in E. Mendes, N.Mosley, (eds), Web Engineering, Chapter 5, New York,Spinger Verlag, 2006

[3] J. R. Miller III, "Professional Decision-Making", New York: Praeger, 1970

[4] J. J. Dujmovic´, "Mixed averaging by levels (MAL)—A system and computer evaluation method" in Proc. Informatica Conf. (in Serbo-Croatian), Bled, Yugoslavia, 1973, paper d28

[5] J. J. Dujmovic´ and R. Elnicki, "A DMS Cost/Benefit Decision Model:Mathematical Models for Data Management System Evaluation, Comparison, and Selection." Washington, DC: National Bureau of Standards,1982, p. 150, No. GCR 82-374. NTIS No. PB 82-170150.

[6] J. J. Dujmovic´, "A method for evaluation and selection of complex hardware and software systems," in Proc. 22nd Int. Conf. Resource Manag. Perform. Eval. Enterprise Comput. Syst., 1996, vol. 1, pp.368–378.

[7] Suleiman H. Mustafa, Loai F. Al-Zoua'bi, "Usability of the Academic Websites of Jordan's Universities"

[8]. Nielsen, J., "Usability Engineering". Morgan Kaufmann, San Francisco, 1994

[9] Holzinger, Andreas. "Usability engineering method" for Software Developers

[10] Nielsen, J. and Mack, R.L. "Usability Inspection Methods" Wiley, New York, 1994

[11] Nielsen, J., (2001), "How to Conduct a Heuristic Evaluation", www.usit.com/papers/heuristic/

[12] Polson, P., Lewis, C., Rieman, J., and Wharton, C., "Cognitive walkthroughs: A method for theory-based evaluation of user interfaces" International Journal of Man Machine http://cat.inist.fr/?aModele=afficheN&cpsidt=5420384

[13] Someren M., Barnard Y., Sandberg J., (1994), "The Think Aloud Method: A practical guide to modeling cognitive processes", Department of Social Science Informatics University of Amsterdam, Published by Academic Press, London.

[14] O. Singnore, "Towards a Quality Model for Websites". CMG Poland Annual Conference, Warsaw, 9-10 May, 2005 URI http://www.w3c.it/papers/cmg2005Polandquality.pdf

[15]. Gilb T., "Software Metrics", Chartwell-Bratt, Cambrigde, Mass., 1976

[16] P.Tripathi, M.Pandey "Towards the Identification of Usability Metrics for Academic Web-sites".

[17] L. Olsina*, D. Godoy, G.J. Lafuente, G. Rossi, "Specifying Quality Characteristics and Attributes for Websites"

[18] L. Olsina, G. Rossi, "Measuring Web application quality with WebQEM", IEEE Multimedia, *9*(4), 2002, pp 20-29.

Software Project Change Management Using Event Calculus

Petros Petrides[1], Andreas Gregoriades[2], and Vicky Papadopoulou Lesta[2]
Department of Computer Science and Engineering, European University Cyprus, Diogenous Str., Engomi, P.O. Box: 22006, 1516 Nicosia-Cyprus.
[1]petros1@eclatent.com, [2]{a.gregoriades,v.papadopoulou}@euc.ac.cy

Abstract - *Software project change is a common problem in project management. Changes in requirements, activities or peoples roles however are usually not free from conflicts, which in some cases if not dealt adequately could bring the whole project to a standstill. Herein, we present a method that examines the impact of change to project duration. The method proposed utilizes two main criteria namely, the degree of dependency among activities and requirements, and the temporal costs associated with the change. The proposed methodology is realised in Event Calculus and elaborated with an example.*

Keywords: Event Calculus, Project Risk Management, Conflict Impact Analysis.

1 Introduction

Despite sophisticated tool support for project management, projects still continue to suffer by budget overruns and a failure to meet user requirements. Various approaches have been proposed as solutions. The capability maturity model integration (CMMI) [6] defines layers of control to help ensure quality and efficient procedures that yield improved software development process. In particular, CMMI's change control, to limit scope creep, and management review, to perform intermediate quality checks on the system, have been shown to be effective in improving project performance. One indicator of quality software is flexibility that allows rapid and cost-effectively modifications of new needs [7]. However, achieving high level of software flexibility has often been slow, inflexible, and time-consuming, even with the aid of sophisticated development tools and methodologies, which oppose the goal of efficient production. Thus, it seemed necessary to examine higher levels of software development flexibility while still meeting budgetary restrictions. These include role management and requirements change management during the software development life cycle (SDLC).

The increased complexity of modern software systems specification makes requirements or role changes an activity of pivotal importance to the successful completion of software systems. Stakeholders demand quality software in reduced costs with the intent to achieve desired competitive edge in the business arena. Both cost and quality are directly linked to effective management of requirements modifications or role changes during the SDLC. Requirements change analysis is translated into change impact analysis that investigates the effect of change to dependent requirements and activities. However, in most of the times changes to requirements or roles during the SDLC are not free from conflicts. If these are not resolved they could lead to inconsistencies in the end product that could end in software failure. Requirements change conflict analysis falls under the requirements management process that is composed of three conceptual phases, namely: requirements dependencies specification, traceability methods, conflict analysis and resolution [5]. The method proposed herein, demonstrates the application of event calculus to identify and resolve conflicts introduced with change. Modifications to project's scheduled tasks are also modeled and accordingly conflicts are identified and resolutions are proposed. The proposed method can monitor the tasks of each team member, and check if the project schedule falls within agreed time limits. Inflicted changes to requirements are also considered so as to assess the extra time introduced.

The paper is organized as follows: next section describes the problem, followed by an introduction to event calculus and prolog. A case study that employs the methods is presented and the key results and conclusions are described.

1.1 Problem Definition

Software project management encompasses knowledge, techniques, and tools necessary to manage the development of software products. It is a methodological approach to achieving agreed output upon specified time frame with defined resources. An important phase during project management is the project specification. The project planning consists of a set of *activities* which may be related to each other via *dependencies*, i.e., action Ai should be implemented after Aj is completed. Conflicts in project management are inevitable. The potential for conflict in information systems development projects is usually high because it involves individuals from different backgrounds and orientations working together to complete a complex task. The objective

of the software project management is to provide to the system designers/managers the detection of conflicts appearing both during project specification and project development. This will aid system designers to adopt appropriate countermeasures to resolve or at least mitigate them.

1.2 Contribution

This work considers projects involving the work of team members. During the development of a project, some tasks are more critical and need to be completed on time. However, the actions of the involving members may cause critical delay that could lead to the failure of the accomplishment of one stage of the project. Also, during the development of the project some changes in the initial specifications of the project may arise resulting in several consequences in the whole structure and specification of the project.

We propose a Project Management tool that monitors the actions of each member and detects possible conflicts that may appear either because of the actions of the users but also because of changes of the specifications of the project. The tool also resolves conflicts by suggesting appropriate modifications in the implementation of the project to guarantee its success. The tool presented herein is based on Event Calculus. In particular, using appropriate events specification and axioms, the tool identifies conflicts upon changes to the characteristics of the project properties. Information that is used during the analysis include: events related to functional project requirements, temporal specification of project characteristics i.e. beginning date of the project, roles, capabilities and actions of project members and modifications to project specifications. Inference rules are used to identify conflicts caused by changes or actions that do not adhere to the project plan. Conflicts resolutions are provided when feasible.

1.3 Related Work

Work by Giorgini et al [5] describes a conflict detection method, composed of three phases namely: modeling, extensional description and verification. During the modeling phase the designer draws the extensional requirements models where every node and edge corresponds to a fact in the formal framework. Then, the reasoning system completes the extensional description of the system using rules and verifies its consistency using constraints. If any inconsistency is detected by the reasoning system, the designer revises the requirements model to avoid or at least mitigate detected conflicts and repeats the formal analysis step. This method uses the Datalog solver which is based on Boolean algebra to detect conflicts. In contrast to this approach the methods described herein is not indented to work solidly on requirements analysis, but also during project implementation where changes are more expensive to be realized and alterations to project schedule more important. EC was also utilized in [5] for policy and specification and analysis. The proposed method transforms policy and system behavior into formal notation which is based on EC. Work by [8] and [4] also use EC as a specialised first-order logic for formalising policy specification and accordingly identify conflicts and propose resolutions in network and systems management. Chomicki [9], similarly use constraints which are policies that prevent a specified action from being performed in a given situation.

2 Background

2.1 Event Calculus

Event calculus was introduced by Kowalski and Sergot as a logic programming formalism for representing events and their effects, especially in database applications [3]. Event Calculus can also be described as a "logical language" from which actions and their effects can be represented using predicates. Therefore, at a specific timeframe, actions that took place and consequences that emerged of those actions, can all be defined, explained and proved using Event calculus. As per another description, it can also be called the "Language of Mathematics" since through logic assumptions the events expressed can be lead to a logic conclusion.

Consider the following example: Maria is hired as a account in a company at May 11 2001, Maria was promoted as a head accountant May 11 2002, Maria left from the position of head accountant at 23 April 2004, can give the following assumptions:

A) Maria was an accountant for a time after May 11 2001.

B) That time ended on May 11 2002,

C) Maria was a chief accountant from May 11 2002 until 23 April 2004.

If the events however were mentioned in the following way: "Maria is hired as an accountant in a company at May 11 2001, Maria left the company as a chief accountant at 23 April 2004" then the assumptions cannot hold; There is a timing gap between the ranking of the person, therefore the promotion cannot be assumed, as other events such as temporally leave could also have happened.

2.2 Logic Programming

Logic programming uses mathematical propositions expressed in computational terms. A solution to problems is divided into two phases: the theorem problem solving, which uses theorems that have been proven in order to give the solution and the programming part, which uses the logical solutions into a program. In logic programming the implications are treated as a base for goal reduction procedures. That comes as a direct opposite to the traditional programming methods,

where first the "IF" statement proceed and then the goal. For instance, the expression: if a=5 and b=6 then a+b=11 is considered to have the same result as the addition, a+b=11, then "a" must be 5 and "b" must be 6.

The proof of the solution is based on two fragments, called *Horn clauses* and *Hereditary Harrop formulas.*

Horn clauses can be written in the form of an implication:

$(p \wedge q \wedge \cdots \wedge t) \rightarrow u$ which means:

"p AND q AND…AND t implies u". A hereditary Harrop formulae is used in logic programming as a generalisation of horn clauses

2.3 Prolog

Prolog is a logic-based programming language developed by Alain Colmerauer, that use the following constructs: An atom that describes a general-purpose meaning, Numbers which can be floats or integers, Variables, denoted by a string consisting of letters, numbers and underscore characters, and beginning with an upper-case letter or underscore and finally, compound terms, comprised of "functor" which is an atom and a number of arguments.

Prolog is a declarative programming language as opposed to procedural programming such as Java. Prolog works by making *deductions* and *derivations* from facts and rules stored in a *database*. The essence of Prolog programming is writing crisp, compact rules. The deductions and derivations, instigated by user-entered queries, are products of Prolog's built-in inference mechanism called *backtracking*.

3 Methodology

The Project management tool is built in EC terms using events and rules. Events related to project implementation, such as the beginning date of the project, actions of members and changes to project specifications are specified using EC constructs. Inference rules are used to backtrack the queries to identify conflicts caused by changes in project plan or actions that do not satisfy project constraints. Conflict resolutions strategies are presented when feasible.

3.1 Describing Events and Relationships with Event Calculus

Consider the following scenario: Two teams are working on car simulation project, which includes both hardware and software. The project is done in parallel development. This procedure has the advantage of allowing the various parts of the project to be worked on simultaneously, while testing can be done after completion of each phase.

Team one consist of Martha, Tim and Thomas. Martha has similar training with Tim and can replace him if necessary. Tim is responsible for the gas system, Martha for the pipeline system and Thomas for the steering system.

The following events take place:

- E1 is an event at which Tim is trained for the gas system. E1 has time 10 March 2004.
- E2 is an event at which Martha is trained for the pipeline system. E2 has time 5 March 2004.
- E3 is an event at which Tim is working for the gas system. E3 has time 21 April 2004.

Recording the recording procedure of the events in the database, note the followings:

- Regarding the databases, updates are addictive; events can be added in the database and no deletions are allowed. In order to delete an event (expressed as a relationship), we add a statement that ends the relationship.
- The chronological events are treated without any particular order; one can add older events after adding more recent events.
- The chronological order of the events can be expressed using the <, >, = symbols. For example, if en event E1 occurred before an event E2 then it can be described as E1<E2.
- Events can also have duration and also a starting point and an ending point.

3.2 Representation

An event concerning the rank of a person is represented using Event Calculus as follows:

X has rank Y for a period E, provided that X has done something at a period that is equal to E.

This means that after an event happened, variable X has rank Y for a chronological period E, which is actually equal to after (E).

In our case, Tim has started working on project at 21 April 2004. Therefore E1 is an event with timing 21 April 2004 at which X has rank Y (Tim is working on project) for the period after E, which that period started April 21 and afterwards. The events E1, E2 and E7 are in the past and their time period will finish when the person has different ranks.

More specifically, the events are expressed using Event Calculus using the following form:

- Time (Event, Time)

To express that event Event happened at time Time.

- Trained(Martha, pipeline system)

To express the person Person was trained for activity A.

- Rank (Person, Time, trained A)

To express the person Person at time Time has been trained for activity A.

So, for example, events E1, E2 can be expressed as follows:

- Trained(Tim, gas system)
- Time (E1 10 March 2004)
- Trained(Martha, pipeline system)
- Time (E2 5 March 2004)

We also use the *Hold* predicate in order express the time periods of the relationships of which the start time of the one is the end of the other. The *Holds(p)* predicate means that a relationship which is associated with P holds for a time period p. Likewise the statement,

- Rank(Tim, working on gas before (E2))

can now be written as

- Holds(before (E2 rank (Tim working on gas))

which means that "Tim holds the rank for working on gas for a time period which is before E2 or in other words which last before E2 starts."

Additionally, the following predicates will also be used:

- Terminates: used to represent termination of an event.
- Initiates: used to represent termination of an event.
- Broken: used to represent identical or incompatible relationships.

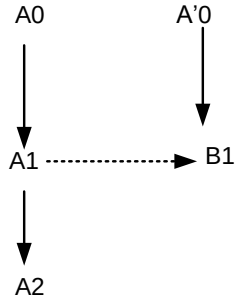
So, for example, Holds(before(e u)) IF Terminates(e u), implies that "An event E holds before time period u, if the event E is terminated at time period u".

4 Example Application of the method in a a Systems Engineering project

We demonstrate our approach with an example drawn from the automotive industry for a vehicle's subsystem. First, the Database of temporal project activities relating to functional requirements and their dependencies is first specified. Next, a second database of actors training is created. Finally, the "event planner" database is specified that records tasks of each actor as scheduled to in the project plan. New events can be added in the database during project execution and after monitoring the project's progress. During project execution changes can be evaluated using queries relating to activities or roles in the project schedule. The tool provides the user with output relating to the impact and feasibility of a change.

4.1 Project Specification

Consider the following model of a project specification depicting *activities* A0, A1, A'0, B1 and their dependencies. The diagram is read as follows: Activity A2 should be implemented before A1 and activity A1 should be implemented before A0, and B1 should be implemented before A'0. The dashed line means that Activity A1 can be replaced by activity B1.



4.2 Actors

There is a set of employees, called *actors* that are involved in the development of the project. An actor may accomplish a specific activity of the project provided he is trained for this task sometime before the implementation of the activity.

So, assume actors P_1, P_2 trained as follows:

- Actor P1 is trained for the implementation of A0, A1, A'0, B1 and A2.
- Actor P2 is trained for the implementation of A0,A1,A'0 and B1.

4.3 Events

Assume a set of events A0-A6, took place during the development of the project. The events are recorded in the database as explained in Section Methodology.

- E0 is an event at which project starts. Date is 10 Feb 2010.
- E1 is an event at which actor P1 implements activity A0. Date is 11 Feb 2010.
- E2 is an event at which actor P1 implements activity A1. Date is 16 Feb 2010.
- E3 is an event at which actor P1 leaves from project for 4 days. Date is Feb 21 2010.
- E4 is an event at which actor P1 returns to the project. Date is Feb 25 2010.
- E5 is an event at which actor P1 implements activity A2. Date is Feb 26 2010.
- E6 is an event at which actor P2 is available for work. Time is Feb 22 2010.

Furthermore, we record in the database using Event Calculus the training of the actors as follows:

- Rank(P1,(before E0),trained A0)) ^ Rank(P1,(before E0),trained A1)) ^
- Rank(P1,(before E0),trained A'0)) ^ Rank(P1,(before E0),trained B1)) ^ Rank(P1,(before E0),trained A2)).

which means

"P1 has training for the activities A0,A'0,A1,B1,A2 at time period before E0".

Also, we record:

- Rank(P2,(before E0),trained A0)) ^ Rank(P2,(before E0),trained A1)) ^
- Rank(P2,(before E0),trained A'0)) ^ Rank(P2,(before E0),trained B1)) .

which means:

"P2 has training for the activities A0,A'0,A1,B1 at time period before E0".

4.4 Conflicts Identifications and Resolution

After recording all events related to the project development we can make queries concerning the temporal feasibility of the project. In particular, we may make queries of the following forms and the system answers appropriately:

- rank_of (P,Y,T): in order to get the rank of actor P at time T. The system will output the action Y that took place at that date.

For example, to find the rank of actor P, at date 16 Feb 2011 we type the question.

"rank_of((p1,Y,(16 Feb 2010))."

The system will search the database for the rank of actor P1 and return the Y which is the rank of P1 at date 16 Feb 2010.

The output answer is

"implements activity A0".

- Activities_of(A, Y): in order to get the dependencies of an action A. This query is very useful when the project manager is interested to change the particular action.

Then, the systems will output list the dependencies of activity A.

- Rank(P,T,trained Y): in order to ask about the training of actor P on activity Y at time T.

For example, if we wish to check whether a change the activity A'0 activity B1 results to no conflicts, we may ask:

- Rank(P1,(before E0),trained A'0)),
- Rank(P1,(before E0),trained B1))

The system outputs TRUE, therefore we can safely assume that actor P1 can implement the change without any conflicts.

4.5 Examples of Queries

4.5.1 Example1

At time period E0, if there is request for change of activity A1 to B1, then there is no conflict at all: This would be the ideal case, as the project has just began and none of the activities have been implemented. So, the following predicate holds:

- Holds (before E0 (P1 implements A0)) FALSE

This mean that "actor P1 has implemented activity A0 before event E0" is FALSE.

4.5.2 Example2

At the time period E1, if there is request for change then:

- Holds (before E1 (P1 implements A0)) TRUE
- Terminates (after E1(P1 implements A0)) TRUE

Replacement part B1 has 2 dependencies-activitys, A0 and A0'.Therefore,

- Initiates (after E1(P1 implements A0')

This means that "Actor P1 starts working on the implementation of A0".

- Terminates (after E1(P1 implements A0')

This means that "Actor P1 has finished the implementation of A0'".

- Holds (after E1 (P1 implements B1))

This means that "Actor P1 has implemented the replacement B1".

In this case, the only conflict would be the case that the actor P1 is not training to implement the change from activity A1 to B1 (and/or their dependencies). Therefore the statement Rank (before E1 AND after E0 (P1 trained for B1)) must also be TRUE.

Checking the ranking of the actor reveals:

- Rank(P1,(before E0),trained B1)) TRUE--→ No conflict

4.5.3 Example 3

At time period E4, date 21 Feb, if there is request for a change of the activities, then there is serious conflict issue, since no one of the actors are available for taking over the change:

- Initiates (after E3 and before E4(P1 implements A0)) FALSE
- Initiates (after E3 and before E4(P2 implements A0)) FALSE

At date 22 Feb, the conflict can be resolved: Actor P2 can replace actor P1 and implement the required parts.

A conflict may appear, when the activity A2 needs to be implemented at time prior of actor P1 arrival:

- Rank(P2,(before E0),trained A2)) FALSE

This means that "Actor P2 is NOT trained for activity A2".

5 Conclusions and Future Work

The work presented herein addresses an important problem in software change management, namely, conflicts analysis and resolution. The method described elaborates on the pressing need for timely delivery of software projects within agreed time limits. The complexity and uncertainty of modern software systems however makes this a challenging task. The statistics of software failures alone highlights the complexity of the problem.

The method described herein provides the mechanism for an effective management of changes in project requirements, activities and roles that in effect impact on overall project completion time. Preliminary results from this work are encouraging. However, the full extent of the method will be realized with its thorough validation. On the same vein, software cost is also another burden to project managers that effectively could bring projects to standstill if not managed adequately. Future activities aim in incorporating software cost estimation models in the method to dynamically assess project cost given changes to requirements, activities and roles during development.

6 References

[1] R. Kowalski., "Legislation as Logic Programs", Springer-Verlag, 1992.
[2] D. Kowalski, Kuehner, "Linear Resolution with Selection Function, Springer-Verlag, 1983.
[3] R. Kowalski, "Predicate Logic as Programming Language", EEE Computer Society Press, 1986.
[4] A. Bandara,E. Lupu, E. Lupu and A. Russo, "Using Event Calculus to Formalise Policy Specification and Analysis", Policies for Distributed Systems and Networks, 2003.
[5] P. Giorgini, F. Massacci, J. Mylopoulos, N. Zannone, "Detecting Conflicts of Interest", Proceedings of the 14th IEEE International Requirements Engineering Conference, 2006.
[6] D. Ahern, A. Clouse and R. Turner, CMMI distilled, Addison-Wesley, 2003.
[7] B. Hugher, Cottetell M. "Software Project Management", McGrahill, 2011.
[8] M.Charalambides, P. Flegkas, G. Pavlou, A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, J. Rubio-Loyola. "Policy Conflict Analysis for Quality of Service Management", 6th IEEE International Workshop on Policies for Distributed Systems and Networks, pp. 99-108, 2005.
[9] Jan Chomicki, Jorge Lobo, Shamim A. Naqvi, "A Logic Programming Approach to Conflict Resolution in Policy Management", Proceedings of the 17th International Conference Principles of Knowledge Representation and Reasoning, pp 121-132, 2000.

Software Safety Engineering Education

David J. Coe, Joshua S. Hogue, and Jeffrey H. Kulick
Department of Electrical and Computer Engineering, University of Alabama in Huntsville
Huntsville, Alabama, USA

Abstract – *This paper describes the Software Safety Engineering Process utilized by students enrolled in the University of Alabama in Huntsville's Software Safety Engineering course. The process consists of an industry-standard software engineering development process augmented to produce the safety-related artifacts as required for certification of a safety-critical product by a regulatory agency. The additional artifacts include Functional Hazard Assessments, Fault Tree Analyses, and Failure Modes and Effects Analyses. Students learn the impact of these new artifacts on the traditional development process as they follow our software safety engineering process to implement a representative safety-critical system, a model railroad controller.*

Keywords: software safety, DO-178B, model railroad, ARP-4761, hazard analysis, fault tree analysis

1 Introduction

Despite the ever increasing dependence on software in safety and mission critical systems, the development processes for safety critical software are frequently absent from the curricula of software engineering degree programs. A recent informal review of degree programs in the US revealed no program dedicated entirely to software safety engineering and only a smattering of courses addressing the topic. The reasons for this include the lack of appropriate text books, the tight linkage to regulatory agencies which is not normally present in academic disciplines, and the absence of good public domain examples ("go-bys") for the multitude of documents that are required for software safety engineering projects.

While safety engineering is a well established discipline with significant support in the form of textbooks, process definitions and professional societies, no such support network exists for software safety engineering. That is not to say there are not emerging standards related to constructing safe software. In fact the multiplicity of different standards is part of the problem of software safety engineering education.

The University of Alabama in Huntsville has begun development of a graduate degree program in software safety engineering within the Department of Electrical and Computer Engineering. Huntsville is well located for such a program with the abundance of organizations such as Redstone Arsenal, NASA, and aerospace companies that develop safety-critical products. This paper outlines the first course in the program, Software Safety Engineering. The course utilizes a *software safety engineering process* derived from industry-standard software engineering and safety engineering processes as well as a representative safety-critical project, a model railroad control system that must be developed to the highest level of design assurance since a train accident would be potentially fatal to humans.

2 Standard software processes

For over forty years, organizations have utilized Waterfall processes for developing large scale software systems [1]. These processes begin by identifying system-level requirements for the product and allocating those requirements to hardware or software components. Once the high-level requirements for the software components have been identified, a Software Development Plan (SDP) is constructed that identifies the artifacts that will be produced along with the development standards, supporting processes, and tools that will be used to produce those artifacts. The standard artifacts include the Software Requirements Specification (SRS), the Software Design Description (SDD), the Software Test Plan (STP), and the Software Test Description (STD) documents. The SDP also lays out a series of milestones for review of these artifacts prior to delivery to ensure that defects are identified and removed as early as possible to reduce cost and improve the quality of the product. Examples of these standard milestones include the Software Requirements Review (SWRR) and the Preliminary and Critical Design Reviews (PDR and CDR). A key component of the SDP are the rough, order-of-magnitude estimates of cost and effort required for development of the product because these estimates dictate staffing decisions and the scheduling of delivery milestones.

Standard Waterfall software development processes, however, are inadequate for the development of safety-critical systems. Systems whose failures pose a danger to life, property, or the environment often require certification by regulatory agencies who examine evidence that the system was developed with a degree of due diligence commensurate with the consequences of

system failure. An integral part of this evidence is the *safety analysis* that identifies potential hazards associated with various system functions and examines the consequences of potential system failures.

The safety analysis is used to determine the degree of due diligence required during development which is called the *Design Assurance Level* or DAL. For example, in the RTCA DO-178B standard used by the Federal Aviation Administration, a system is designated as DAL A if a software fault may result in a catastrophic failure (i.e., a crash, multiple deaths) [2]. DALs B through D are reserved for systems whose failures result in less severe consequences. For instance, a system is designated as DAL B if software faults may result in a hazardous failure (i.e. significant reduction on performance or safety of system), and a system is designated DAL D if a software fault results only in a minor failure (noticeable failure with little to no safety impact).

The DAL assessment has a major impact on system cost and delivery schedule because of the additional supporting evidence that may be required by the certification agency. For each assurance level, DO-178B requires evidence that specific sets of process objectives have been completed with the number of objectives required increasing with each increase in the design assurance level. For a system assessed at DAL C, one must supply evidence that 57 objectives have been satisfied while a system assessed at DAL A must satisfy 66 objectives [2]. Furthermore, the rigor with which a given objective must be addressed varies from DAL A to DAL E. For example, there are DO178-B objectives (A7.5 – A7.7) that address structural coverage testing requirements. Modified condition-decision coverage (MC/DC) testing is mandated for DAL A systems while only statement and decision test coverage are required for DAL B and DAL C systems. Consequently, a *software safety engineering process* produces more documentation, at significant additional cost, than traditional software engineering processes.

The safety analysis, however, cannot be a one-time input to a traditional software engineering process since subsequent design and implementation choices can increase the likelihood of system failure. For example, in the systems requirements process, safety analysis may only be performed on the system functions that have been identified at that time whereas safety analysis of the design cannot occur until a design has been produced later in the development cycle. Thus, safety analysis must be an iterative activity that is integrated throughout the development process which is one fundamental difference between a traditional software engineering process and a software safety engineering process. In the following section we begin by discussing a standard safety analysis process and then describe how we have integrated safety analysis into our *software safety engineering process* as illustrated in Figure 1 below.

3 Software safety engineering process

The SAE ARP 4761 standard defines the requirements for system and software safety analysis that must be completed including an initial Functional Hazard Assessment (FHA) of the system before starting hardware/software development [3]. The FHA identifies system functions and the failure effects and conditions if the system functions were to fail. The FHA also determines the severity for the failure of each system function. For example, if the system requirement were "the system shall schedule low priority trains onto the sidings", then a failure to perform that function is a hazard that may result in a crash.

A Fault Tree Analysis (FTA) is then performed on the top-level functions identified in the FHA, breaking them down into a boolean tree of lower-level events with the lowest level named the basic events. Consider the previous example of a system hazard – failing to schedule a low priority train onto a siding. Mechanical switch failure is one example of a low-level basic event that could be the source of that hazard.

The failure effects, severity, and modes of these basic events are described within the Failure Modes and Effects Analysis (FMEA). After the completion of the FHA and FTA, the system functions are allocated to hardware or software. Depending upon the severity of those functions, independent design assurance levels (DALs) are determined for hardware and software via System Safety Analysis (SSA). For developers to accurately identify the design assurance level and subsequently conform to the objectives associated with that level, a rigorous, planned process is required since the DO-178B only provides guidance on objectives to be completed without imposing specific activities required to meet those objectives [4].

Figure 1 below shows our approach for safety-critical software development which consists of DO-178B for software development integrated with ARP-4761 for system safety. The lists of artifacts indicate when within the life cycle the artifacts were generated. Those artifacts specific to safety analysis have also been identified. End-to-end traceability is a key requirement of DO-178B.

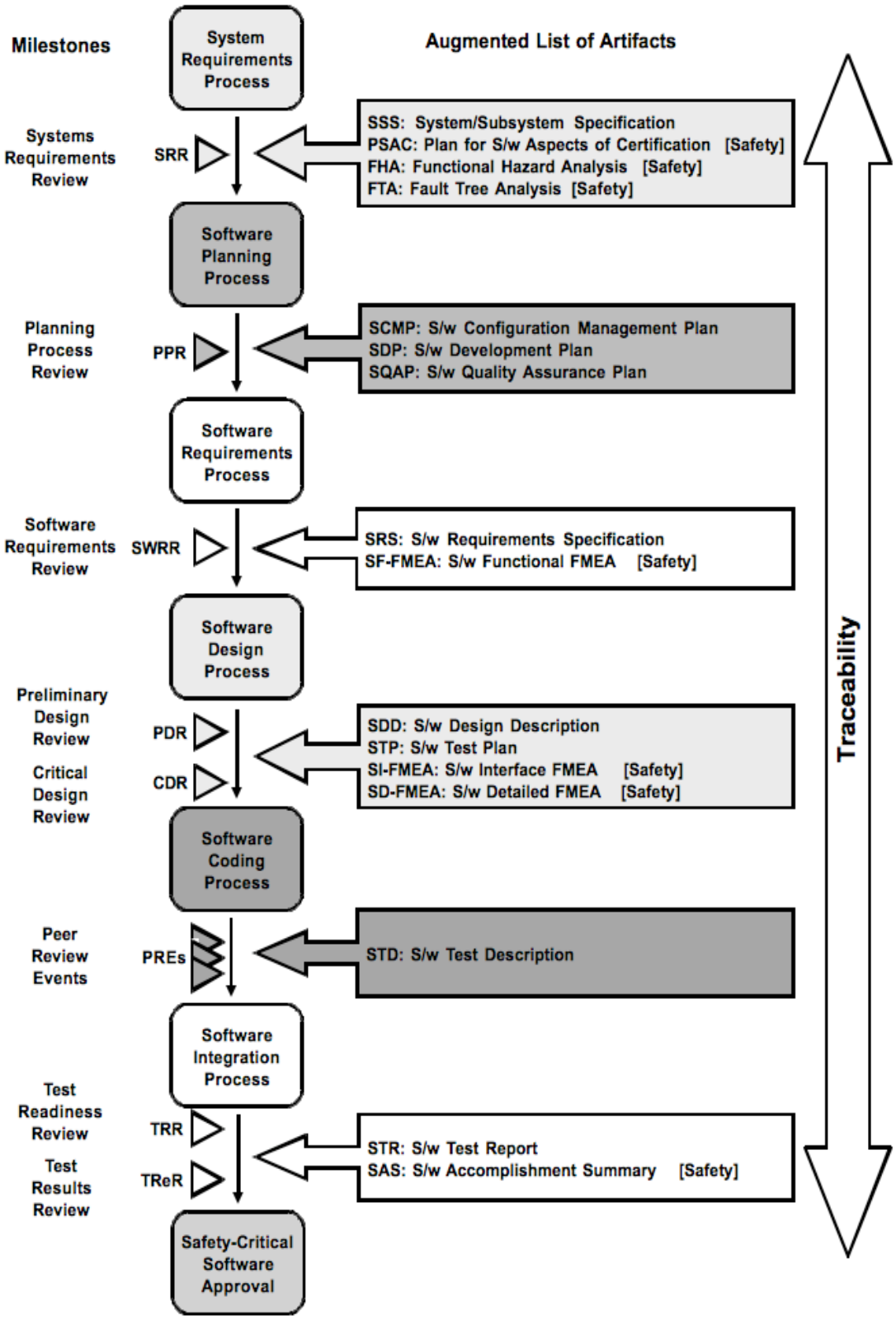


Figure 1 – Software safety engineering process with safety artifacts identified.

Table 1 – Safety and software engineering artifacts

Safety Artifacts (ARP-4761)	**Integral Process Artifacts (DO-178B)**	**S/w Engineering Artifacts (DO-178B)**
Functional Hazard Assessment (3.2)	Verification Artifacts (11.13, 11.14, 11.17)	Planning Artifacts (11.1 – 11.5)
Fault Tree Analysis (4.1)	Configuration Management Artifacts (11.15, 11.16, 11.18)	Development Standards (11.6 – 11.8)
Failure Modes and Effects Analysis (4.2)	Quality Assurance Artifacts (11.19, 11.20)	Development Artifacts (11.9 – 11.12)

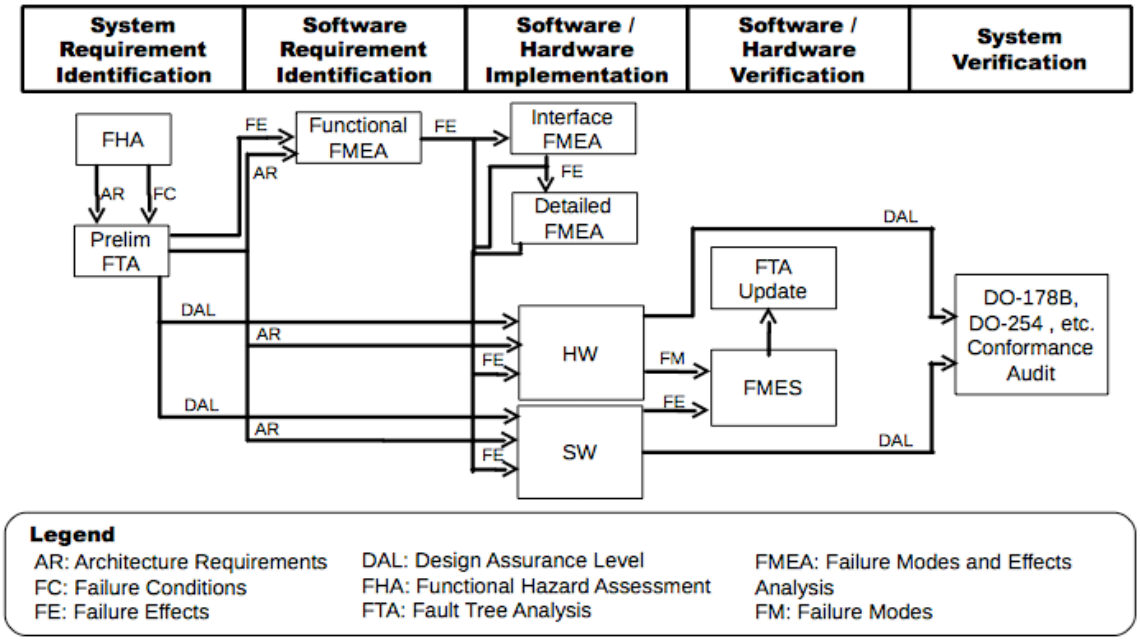


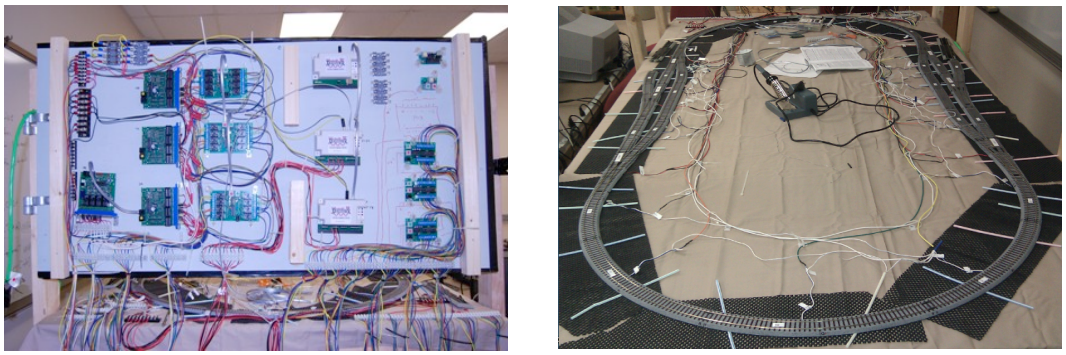
Figure 2 – Software/safety engineering process flow

The primary artifacts of interest in our Software Safety Engineering course come from several different categories, as shown in Table 1. The safety artifacts developed are based upon the SAE ARP 4761 standard, which provides guidance for FHAs, FTAs, and FMEAs in the sections shown in parentheses. DO-178B was used for development of the integral process artifacts and software engineering artifacts. The applicable DO-178B sections for the artifacts are given in parentheses.

The software engineering/safety engineering hybrid process flow integrates safety analyses in the initial determination of a DO-178B DAL through the FHA and preliminary FTA. The FMEAs are then developed simultaneously with the software requirements, design, and coding artifacts. By conducting FMEAs in the development phases, the safety analyses cover all levels of the software design and are relevant to the real system's behavior and hazards [5]. The FMEAs also provide for an indirect check on the correctness of the original software requirements, since faulty design and code are derived from faulty requirements [6]. By incorporating safety analysis throughout every phase of the software life cycle, the design team substantiates their initial DAL selection. The flow of this process is shown in Figure 2, based upon a modified hybrid of DO-178B processes and ARP-4761 diagrams for the safety assessment flow. The overview shows how the course process guarantees the software safety and software engineering activities are coalesced into a single efficient development process.

4 Model railroad system project

To fully appreciate the differences between a standard software engineering process and a software safety engineering process targeted at system certification, students must follow our software safety engineering process as they develop a safety-critical product, producing the artifacts required for certification as the project evolves. The teaching platform of choice for our Software Safety Engineering course is a model railroad project. As shown in Figure 3, the model railroad layout consists of an oval track with two sidings for passing locomotives and two dead-end spurs. A safety system is required since multiple trains traveling around the oval at different speeds and in different directions may collide.



(A) (B)

Figure 3 – Model railroad setup (A) Control logic and (B) Track layout

The primary safety system for the model railroad is a scheduling system managed by the off-the-shelf *TrainController Bronze* Scheduling Software from Freiwald Software [7]. The track layout is divided into twenty four, individually powered segments. Digitrax logic boards monitor conductive detectors within each segment to determine the segment where each train is located, and this position information is forwarded from the logic boards via USB cable to the PC-based scheduling software [8]. Similarly, commands from the scheduling software are passed via the logic boards to specific locomotives and railway switches.

Before a train may travel from one segment to another, it must reserve the next segment of track. If the next segment is available, the scheduling software sets the appropriate railway switches and allows the train to continue traveling onto the next track segment towards its destination. If the next segment is currently occupied, then the train must either stop and wait for the segment to become available or the scheduler must set a switch that diverts the train onto a siding or spur so that an on-coming train may pass.

From a teaching perspective, the model railroad project has a number of advantages. First, the baseline system is relatively low cost (approximately $5000) which is advantageous to universities. The model railroad system can also be reconfigured at modest cost to increase or decrease complexity as needed. Another important factor is that the safety hazards associated with the model railroad are easily understood and can be safely illustrated using the model railroad system. Such hazards include head-on-collisions or derailments due to incorrect railway switch settings.

5 Software safety engineering tools

A critical educational aspect of the course is the hands-on experience that students gain as they apply commercial-grade software engineering tools to generate and manage the artifacts required for safety certification. Safety certification requires the development of and strict adherence to Software Quality Assurance (SQA) and Configuration Management (CM) plans that span the entire development life cycle. These plans must address end-to-end requirements traceability, document and coding standards, structural coverage testing, and configuration and change management.

Students enrolled in the Software Safety Engineering course make use of LDRA's Testbed and TBrun for extensive static and dynamic analysis of their own code [9]. As part of the static analysis procedure, the code is audited for compliance with industry-standard, best-practice coding standards such as MISRA-C, CERT C, and JSF C++ AV standards. The LDRA tool also performs additional static analyses including code complexity analysis, reachability analysis, and data-flow analyses. Dynamic analysis of the code allows students to execute and test their code to determine the degree of structural coverage achieved -- a key feature since different design assurance levels require different levels of structural coverage such as statement coverage, branch coverage, or modified condition/decision coverage (MC/DC).

Students also utilize a combination Wind River's Simics and VxWorks tools for early on-target testing [10]. Simics is used to simulate the target platform itself. VxWorks is a real-time operating system (RTOS) used on the actual target. With student software running under VxWorks on the Simics simulator, the students gain early insight on the performance of their software on the target platform but in a simulated environment where it is fully accessible for analysis.

6 Results

As of this writing, the students have completed the system requirements process and software planning process, and they are currently working on the software

Table 2 – Functional hazard assessment results [12]

Function	Failure Effect	Severity (ARP 4761)
Locate train on track	Train location is unknown and trains may collide	Catastrophic
Power down on emergency stop	Trains cannot be stopped and may collide	Catastrophic
Accept/Verify track configuration	Trains cannot be ran on track	Minor
Accept/Verify train configuration	Trains cannot be ran on track	Minor
Control turnouts on plant	Trains cannot be safety directed and may collide	Catastrophic
Accept/Verify train schedules	Trains cannot be ran on track	Minor
Control trains according to schedule	Trains operate with limited or no control and may collide	Catastrophic

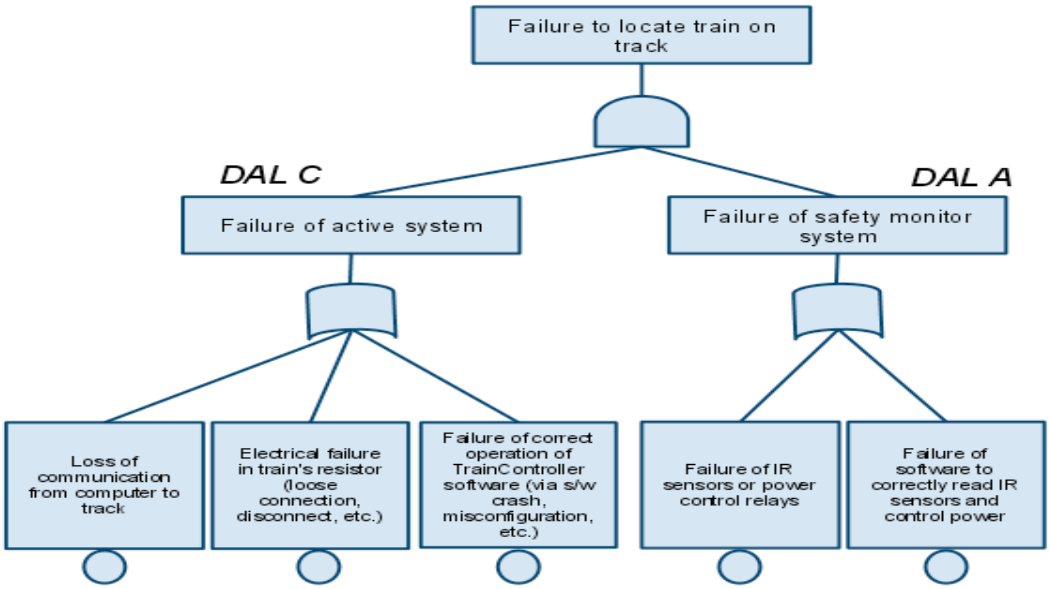


Figure 4 – Fault tree analysis for "Locate Train on Track" function [12]

requirements process artifacts. Since they have not completed the development of the full model railroad system, only initial results are available. However, even the initial results provide an insight into the advantages of our software safety engineering process.

The students set out to design the complete train scheduling system around the off-the-shelf Scheduling Software. The students' initial system requirements specification and FHA identified seven system-level functions of the Scheduling Software and also identified the failure effects of those functions and the severity classification of the failure effects. The system functions, failure effects, and severities are shown in Table 2.

Using the FHA analysis, the students ascertained that the Scheduling Software must meet DO-178B level A, since the worst case failure of the Scheduling Software would lead to the catastrophic event of two trains colliding. After evaluating the requirements for DO-178B level A, the students decided that developing the model railroad system to satisfy DAL A would be extremely difficult due to the complexity of the Scheduling Software and the lack of source code for that software. As a result of their safety analysis, students devised an *amelioration plan* in which they would develop a *Safety Monitor* system to handle the safety-critical aspects of the system independently of the Scheduling Software.

The *Safety Monitor* is responsible for continuously monitoring train position and cutting power to the tracks to prevent catastrophic events such as collisions from occurring. The Safety Monitor system consists of CTI Electronics optical sensors for track segment entry/exit detection [11], power control modules, and a single monitoring software procedure. Unlike the Scheduling Software which may be thousands of lines of code, the Safety Monitor only has to observe adjacent track segments and shut off power if a collision is eminent. As a result of this simplicity, students determined that it would be easier to develop the Safety Monitor to satisfy

DO-178B DAL A standard than to demonstrate that the more complex, off-the-shelf Scheduling Software satisfies DAL A.

Students demonstrated the efficacy of this Safety Monitor approach through the FTA of the system functions. As illustrated in Figure 4, a catastrophic severity function (level A) is comprised of the "active system" (Scheduling Software) providing the function capability at DO-178B level C and a Safety Monitor providing the safety-critical protection of the system at DO-178B level A. A failure will only occur through the failure of both the Scheduling Software and the Safety Monitor. By introducing the Safety Monitor system, the Scheduling Software's required assurance level was dropped from A to C, drastically reducing the number of DO-178B objectives and activities that must be completed with independence.

7 Conclusions

As software continues to become an integral component of more and more products whose failures may prove catastrophic, the demand for instruction in software safety engineering processes will increase. One of the challenges in offering our software safety engineering course has been the general lack of understanding among software developers regarding safety analysis and how a software safety engineering process differs from standard software development processes. It has become clear that while there are numerous software engineering courses and texts that discuss standard software processes and specific development activities such as requirements elicitation and testing, there are few texts that discuss the integration of safety analysis into the software development process and how safety certification requirements impact the artifacts that must be delivered. Our use of the model railroad project in our Software Safety Engineering course has proven to be a very effective means of conveying the nuances of software safety engineering to our students when there are few other resources readily available.

8 Acknowledgements

The authors would like to acknowledge the following contributions. George Petznick for assistance with wiring. James Norris of Redstone Arsenal Model Railway Club for model railway discussions. LDRA and WindRiver for software donations. Todd White from FAA Consultants for document templates. Steve Hosner for guest lecture on Fault Tree Analysis.

The authors are especially grateful to Elise Haley, Joshua Hogue, Michael Gallaher, and Hunter Stinson for granting us permission to include a portion of their FHA and FTA in this manuscript.

9 References

[1] W. R. Royce, "Managing the Development of Large Software Systems," *Proc., IEEE WESCON,* 26, 1970, pp. 1-9.

[2] RTCA DO-178B, Internet: http://www.rtca.org/

[3] SAE ARP-4761, Internet: http://www.sae.org/

[4] C. Bertrand, C. Fuhrman. "Towards defining software development processes in DO-178B with openup." *Canadian Conference on Computer and Electrical Engineering*, 2008.

[5] K. Allenby, T. Kelly. "Deriving Safety Requirements Using Scenarios." *Fifth IEEE International Symposium on Requirements Engineering*, 2001.

[6] A. Arkusinski. "A Method to Increase the Design Assurance Level of Software by Means of FMEA." *The 24th Digital Avionics Systems Conference*, 2005.

[7] "Friewald Software." Internet: http://www.friewald.com.

[8] "Digitrax." Internet: http://www.digitrax.com

[9] "LDRA Software Technology Homepage." Internet: http://www.ldra.com.

[10] "Wind River Simics." Internet: http://www.windriver.com/products/simics/.

[11] "CTI Electronics." Internet: http://www.cti-electronics.com.

[12] E. Haley, J. Hogue, M. Gallaher, and H. Stinson, Safe Train Project Documentation, unpublished.

Project Success as an Evolving Concept

Kadir Alpaslan Demir
Department of Computer Engineering
Turkish Naval Academy
Tuzla,Istanbul, Turkey
kademir@dho.edu.tr

ABSTRACT
Identifying project success rate and project success factors has been an interest for researchers quite a long time. Traditionally, project success is measured based on performance in three criteria: Cost, schedule, and delivered functionality. Various studies included other factors. In addition, there are studies analyzing project success from the perspectives of different stakeholders. Currently, there are no universally accepted criteria and definition for project success. Without a well-established definition, the researchers have the risk of mislabeling a project as success or failure or vice versa. Therefore, in this paper, rather than trying to provide a universal definition for project success, we provide another way of thinking such as defining success from each stakeholder's perspective. First, we review and discuss the criteria used in the evaluation of project success and the perspectives of various project stakeholders. Finally, we explain a simple tool called project success analysis matrix (PSAM) that can be used to determine whether a project may be considered success or not from a specific stakeholder's perspective.

Categories and Subject Descriptors
K.6.1 [**Project and People Management**]: Project success, Project success evaluation D.2.8 [**Metrics**]: Software metrics, Process metrics, Project metrics

General Terms
Management, Measurement.

Keywords
Project Success, Software Project Success, Software Metrics, Project Metrics

1. INTRODUCTION
Numerous studies have been conducted on project success, project success factors, defining them and measuring them. In addition, project failures have been a huge interest. There are also studies on that topic too. Traditionally project success is measured based on the performance on three criteria: Schedule performance, budget performance, and delivered functionality. These factors are sometimes called "iron triangle". One of the famous studies that use these factors is Standish Group`s CHAOS study. They first conducted the study in 1994 and they repeated the study in the following years (The Standish Group, 1994). According to the 1994 study only 16% of IT projects were successful. Project success was defined as delivering the product on time within budget and with the desired functionality. The latest report was published in 2009 and it reported that only 32% of IT projects were successful. Based on this latest study 68% of IT projects were either challenged or cancelled. El Emam and Koru conducted another interesting study in 2005 and 2007 (El Emam and Koru, 2008). They report that talk of a software crisis may be exaggerated and the rate of IT project failures, including cancellations and delivered projects with unsuccessful performance, is only between 26% and 34%. These numbers are quite different from each other. Furthermore, several researchers provide criticisms on the results in CHAOS study (Glass, 2002; Glass, 2005; Jorgensen and Molokken-Ostwold, 2006).

Standish Group's CHAOS report defines a successful project as "it is completed on time and within budget, with all features and functions as specified" (The Standish Group, 1994). This widely used definition of project success, "iron triangle", is not enough. Let us consider some cases. What should we label a project when it has a %40 cost overrun, 20% schedule overrun but the developers delivered all the functionality and the customer is quite satisfied with the product? Should we label it a success or a failure? What sould we label a project when there is only 5% cost overrun, 7% schedule overrun, 100% of the promised functionality is delivered? Another interesting and real world case might be a project on time, within budget with full functionality and the customer decided not to use the product and shelf it. Let me add another real world case study that I investigated. A large software project had cost and schedule overrun. The developers had to deliver more functionality than stated in the initial baseline. I have interviewed the project manager of this large project. The project manager considers the project quite a success. The project manager also mentioned that the cost was not a priority in this project. Because the customer organization needed the product to continue its operations and basically the organization needed the product to exist. Therefore, schedule performance was far more important than cost performance. The project manager explained why the project should be considered such a success when it had cost and schedule overrun. Because the product is successfully in use after many years. Isn't that an evidence of success? With such cases, the reported figures on project success and failure rates may be misleading. Furthermore, the traditional definition misses some of the important perspectives such as the practitioners' perspectives. Robert Glass raised the issue with an article named "Evolving a new theory of project success" in Communications of the ACM (Glass, 1999). Glass was emphasizing the doctoral research of Lindberg. Glass was quoting Lindberg and saying "A new theory of software project success may be necessary." Lindberg (1999) and Procaccino et. al. (2005) analyzed software project success and failure from the practitioner's perspective. Sometimes, it is possible that even when the project is cancelled, it still may be considered as success by the developers. The lessons learned from the project may provide important input for the next project. Another important criterion missing from the traditional definition is user/customer satisfaction. The users' perception of the project success may change over time based on their experience with the product. Agarwal and Rathod (2006)

investigated the notion of success for a set of internal stakeholders: Programmer/developers, project managers, and customer account managers. They report that some software practitioners consider customer happiness, satisfaction, and project specific priorities as important criteria for determining success.

Pinto and Slevin (1998), points out the difficulty of developing a method for analyzing and predicting the success or failure of projects. One of the reasons is that successes and failures are in the eyes of the beholder (Pinto and Slevin, 1998). They also state that until we have a set of criteria that have some generally accepted basis for assessing projects, we have the risk of mislabeling projects as success or failure. While I agree that successes and failures are in the eyes of the beholder, I do not think we will have a true objective set of criteria for assessing success. In addition, De Wit (1988) states that "one can objectively measure the success of a project is somewhat an illusion". We may use some defined set of project success criteria to assess project performances. We may comment on these figures but we should not be easy on labeling whether they are successes or failures. In this paper, I argue that the current set of criteria won't be enough to answer the question whether the project is successful or not. And, the success or failure of a project will remain subjective. The solution lies with defining success for each stakeholder in a project.

The rest of the paper is organized as follows. Section 2 reviews the factors used in assessing project success. Section 3 identifies different stakeholders and their views on project success. In the following section, we explain a simple tool called project success analysis matrix (PSAM) that can be used to evaluate project success. Finally, we conclude the paper with possible future work.

2. CRITERIA USED IN MEASURING PROJECT SUCCESS

The notion of project success has been changing or better put evolving in years. During 1960's, the notion of project success evolved around three factors: Schedule, budget and project performance (Avots, 1969; Gaddis, 1959). If the project is completed on time and within budget and performing what it was supposed to perform, it is considered a success. During late 1980's, another criterion was used in measuring success: Customer or client satisfaction. It was realized that even though the first three criteria were satisfied, the customer might still not be happy with the product. With this finding, ways to measure customer satisfaction were developed. During late 1990's, studies showed that practitioners might have a different perspective on project success. They may view the project as a success even though the project is canceled or unsuccessful when measured using traditional criteria.

There is another criterion from an organization's perspective: Conformance to the long-term goals. Organizations have long-term goals such as making profit, increasing the market share, improving productivity etc. Organizations need and undertake projects for a variety of purposes during their lifetime. Examples include development of a new product, increasing the effectiveness of a particular department, establishing a new policy or a protocol. Some of these projects may fail. However, lessons learned during the project may help to the organization to achieve long-term goals. For example, some technology developed, a discovery made, or lessons learned during a failed project may help another project and the organization to achieve its long-term goals.

Time is an important factor in the notion of project success. The time when the project success is measured should be carefully identified and the result should be correctly interpreted. Project success changes over time (Pinto and Slevin, 1988). For example, customer satisfaction may change when similar products are introduced to the market or the project hardware starts to deteriorate. In software projects, maintenance costs may increase or the product may become unstable due to changes in the environment, upgrades or bug fixes. The result is decreased customer satisfaction over time.

Up to now, we identified 7 criteria and factors that are closely related to project success evaluation:

1. Schedule performance
2. Budget performance
3. Delivered functionality
4. Customer/Client/User satisfaction
5. Lessons learned / Knowledge acquired
6. Conformance to long-term goals
7. Time

As it is argued, the notion of project success is evolving. The success criteria list may include other specific success criteria for different stakeholders such as practitioners' and third parties' views. In addition, it is possible that new criteria may be discovered to be used in measuring project success. For example, system of systems engineering is becoming a hot topic and attracting researchers in recent years. Following the trend, it is possible to have a candidate project success criterion, which may be the ease of integration to systems. As the human needs and the business environment evolves, the notion of project success will evolve and new criteria will be used to assess project success.

3. PROJECT STAKEHOLDERS AND PROJECT SUCCESS

Project Management Institute's Project Management Body of Knowledge (1996) defines project stakeholders as "individuals and organizations who are actively in the project, or whose interests may be positively or negatively affected as a result of the project execution or successful project completion". For every project, the stakeholders need to be clearly identified and their needs must be investigated. Then, measurable success criteria should be set for stakeholders. Otherwise we won't able to determine whether the project is a success or not. Let's analyze the stakeholders and how the factors are related.

Customer: Every project starts with a need and behind this need there is a customer. The customer is the main stakeholder in a project. The project team should work closely with the customer in clarifying the needs and turning the needs into clear project requirements. Also, every need has implicit properties such as a time frame and associated cost. Outside the time frame and the allowable cost, the need has no meaning. After the project is deployed, customer satisfaction becomes an important factor for the determination of success. Project maintenance, introduction of similar products and changes in the environment are among the

issues affecting customer satisfaction. In addition, customer experience with the project is another factor that affects the perception of project success. Thus, success changes over time.

Organization: The organization is another important stakeholder. The organization is the hosting entity of the project development. A project may be developed under a department within an organization or the organization itself may be the developer. The organization has always other goals such as survival, making profit, expanding, or increasing its market share. The success of a project may help the organization reaching these goals. At least, the success of a project helps the organization with the image. The more the project success is inline with the long-term goals of the organization, the more it is valuable.

Third Parties: Other than the customer and the organization, there are third parties that will be affected by the project outcome. A mitigating organization between the customer and the organization, other companies or organizations, and the government are among the third parties. A successful product may decrease the market share of another company. The environment the project affects may require the introduction of new regulations, standards or laws. Or the project may require compliance with the current standards, policies, regulations etc. Therefore, the government may become a stakeholder. Factors related to third parties should be identified when they are necessary for determining success.

Practitioners: The project developers are the practitioners and generally a secondary stakeholder to the project. The success of a project is a good motivator for the developers. However, the practitioners may also have different motivators and an assessment of the project. Even though the project may be viewed as a failure with consideration of previously stated factors, the practitioners may consider the project as a success (Procaccino et. al., 2005).

To evaluate project success, we have to identify the stakeholders and related success criteria. Satisfying every stakeholder may not be possible because of conflicting interests. To determine whether the project is a success or not, it is crucial to develop a matrix consisting of the stakeholders and the criteria that are important for them. The stakeholders should be prioritized, since it may not be possible to satisfy each of them. In addition, the criteria should be prioritized too. The matrix doesn't have to be a complete one; however it should at least include the important stakeholders and criteria important for them.

4. PROJECT SUCCESS ANALYSIS MATRIX (PSAM)

A project success analysis matrix (PSAM) is a matrix that includes project success criteria for each stakeholder. This matrix may be different for each project while most criteria may be similar in many projects. PSAM may be used as a guide during the development of the project. Important project decisions should be made in such a way that it helps to achieve project success criteria based on prioritization. Furthermore, with such data available, researchers will be able analyze project success and failure factors in detail. PSAM is developed by the developer organization with the help of stakeholders to provide a project success definition for the specific project at hand. When a specific set of criteria is satisfied in PSAM, the project may be considered successful from the corresponding stakeholder's perspective. If the developers do not develop PSAM during the project, researchers may develop it by interviewing project stakeholders after the project is completed.

The best time to develop a PSAM is at the beginning of the project as soon as the stakeholders and their concerns are identified. However, PSAM may be developed any time as an aid to determine whether the project is successful from a specific stakeholders' perspective. Even researchers or analysts may develop PSAM for a project to conduct a project success analysis in a systematic way.

Project success analysis matrix is developed as outlined below:

1. Write down the date of the analysis.
2. List the stakeholders and prioritize them.
3. Write down the stakeholders in the left row of the matrix in a descending order based on their priority.
4. For each stakeholder, identify the success criterion or criteria.
5. Prioritize the criteria (if there are more then one criterion).
6. Write down the criteria in the top column of the matrix starting with the most important criterion.
7. If a criterion is important in determining success for a specific stakeholder, put a checkbox in the corresponding cell. Otherwise mark the cell with "don't care".
8. Repeat step 7 until all cells are filled with a checkbox or "don't care".

A sample PSAM is given in Table 1.

5. CONCLUSIONS

As it is discussed here, project success is an evolving concept; thus it is a moving target. Trying to establish a set of universally accepted project success criteria seems as an endless effort. De Wit calls it an illusion to objectively measure the success of a project (De Wit, 1988). Pinto and Slevin (1988) concluded that project success is a complex and often illusory construct. Therefore, we have to look at the problem in a different way. Rather than trying to find the universal objective set of criteria for project success, we have to define project success for a specific project. Here, we also provide a simple tool called project success analysis matrix (PSAM) to help with this. This matrix provides an overall view of project success for each stakeholder with a specific set of criteria. Only after such data available, it is possible to conduct studies on the success and failure rates of projects in various industries. Unless we define what constitutes as project success for the stakeholders, we will continue to mislabel projects and disregard the views of some stakeholders. Furthermore, we will continue to mislead practitioners on the status of project-based industries.

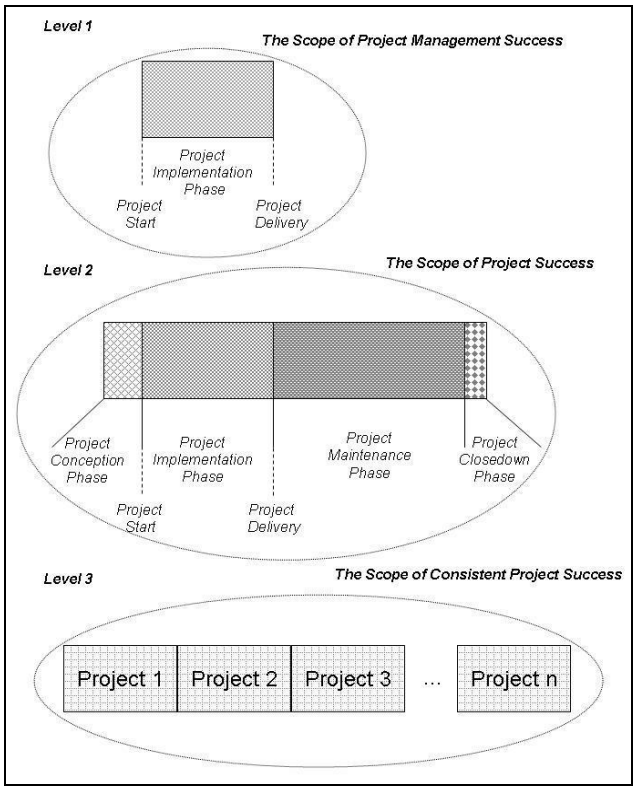


Figure 1. Scope of Success at Different Levels [Taken from (Demir, 2008)]

Figure 1 shows the scope of success at different levels. The scope of project success includes project conception, implementation, maintenance, and close down phases. These are the phases for a project life cycle. However, the scope of project management success includes only the project implementation phase. Project success criteria such as budget, schedule, and delivered functionality performance are the result of project management performance during project implementation. Therefore, these classical criteria are in fact criteria for project management success.

Furthermore, in our research (Demir, 2008), we developed a measurement tool to measure software project management effectiveness achieved during project implementation. We have conducted in-depth analysis and measurements regarding project management aspects on 16 software projects from Europe and North America. Our findings indicates that only half of the variation in project success ratings can be explained with project management effectiveness achieved during project implementation. The other half of the variation in project success ratings can be explained with other factors related to the project life-cycle phases outside the implementation phase. It is clear that the classical criteria used to measure project success are not comprehensive.

As a result, we have to establish a new way of thinking on project success. Rather than evaluating project success with classical criteria, we have to define project success for the project at hand from each stakeholder's perspectives and evaluate project success based on the definition.

Future work will include conducting case studies using PSAM. Furthermore, another study might be how much the widely used criteria can be attributed to overall success of a project.

6. ACKNOWLEDGEMENTS

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of any affiliated organization or government.

7. REFERENCES

Agarwal, N., Rathod, U., (2006). "Defining success for software projects: An exploratory revelation", *International Journal of Project Management 24* (2006) pp. 358–370.

Avots, I., (1969). "Why Does Project Management Fail?", *California Management Review,* 1969, vol. 12, pp. 77-82.

Demir, K. A., (2008). Measurement of Software Project Management Effectiveness, PhD. Dissertation in Software Engineering, Naval Postgraduate School, December 2008.

De Wit, A., (1988). "Measurement of Project Success", *International Journal of Project Management*, Vol. 6, Issue 3, August 1988, pp. 164-170.

El Emam, K., & Koru, G. (2008). "A Replicated Survey of IT Software Project Failure Rates." *IEEE Software*, 2008.

Gaddis, P. O., (1959). "The Project Manager", *Harvard Business Review,* 1959, vol. 35, pp. 32-35.

Glass, R., (1999). "Evolving a New Theory of Project Success", Communications of the ACM, Vol. 42, No. 11, November 1999.

Glass, R., (2002). "Failure is looking more like success these

days", IEEE Software, Jan/Feb 2002.

Glass, R., (2005). "IT Failure Rates-70% or 10-15%?", IEEE Software, May/June 2005.

Jorgensen, M., & Molokken-Ostwold, K. (2006). "How Large Are Software Cost Overruns? Critical Comments on the Standish Group's Chaos Reports." *Information and Software Technology, 48(4)*, 297-301.

Lindberg, K. R., (1999). "Software developer Perceptions About Software Project Failure: A Study", The Journal of Systems and Software, Vol. 49, Number 2/3, 1999, pp. 177-192.

Pinto, J. K., Slevin, D. P., (1988). "Project Success: Definitions and Measurement Techniques", P*roject Management Journal,* Vol. XIX, Number 1, February 1988, pp. 67-71.

Pinto, J. K., Slevin, D. P., (1998). "Critical Success Factors", *Project Management Handbook,* Jossey-Bass Publishers, San Francisco, 1998, pp. 379-395.

PMI Standards Committee, (1996). *A Guide To The Project Management Body of Knowledge 1996 edition,* Newton Square, PA, USA, 1996.

Procaccino, J. D., Verner, J. M., Darter, M. E., Amadio, W. J., (2005). "Toward predicting software development success from the perspective of practitioners: an Explanatory Bayesian model", *Journal of Information Technology,* Vol. 20, 2005, pp. 187-200.

The Standish Group, (1994). *The Standish Group Report: Chaos*.

Table 1. A Sample Project Success Analysis Matrix (PSAM)

Date: 01.01.2011	**Project Success Criteria**					
Stakeholders	**Cost**	**Schedule**	**Delivered Functionality**	**Conformance to long-term goals**	**Lessons Learned**	**….**
Customer/Client	√	√	√	Don't Care	Don't Care	
User	Don't Care	Don't Care	√	Don't Care	Don't Care	
Developer Organization	√	Don't Care	√	√	√	
Practitioners	Don't Care	Don't Care	√	Don't Care	√	
Organization using the services of the client	Don't Care	√	√	Don't Care	Don't Care	
Consultants to the developer organization	Don't Care	Don't Care	Don't Care	Don't Care	√	
….						

On the shortest path to satisfy software projects' core requirements

Jie Liu[1] and Ted Beers[2]
[1]Department of Computer Science, Western Oregon University, Monmouth, Oregon, USA
[2] Hewlett-Packard, Corvallis, Oregon, USA

Abstract - *In the earlier days of software engineering history, a software project meant that the team must implement every requirement. Nowadays, a company has many pieces of software carrying out countless tasks. In addition, many software systems communicate to with the outside world following some well established standards and consist of reusable and expendable components such as APIs and web services. In this case, to fulfill some core set of requirements does not always means to develop complete new software systems. Often, the core set of requirements can be satisfied by utilizing existing infrastructure or software systems. Clearly, taking such an approach is much more desirable in most of the cases because the costs for development, testing, user training, and maintenance are much lower. In addition, the time to market is much shorter.*

In this paper we propose a very different from the traditional software development approach where a software project means to specify tools, infrastructure, and ecosystem so the development can take place. We suggest that before build a new software system, we should look hard to identify existing software solutions to fulfill part, of not all, of the set of core requirements. The approach is demonstrated by providing details of three successful software projects.

Keywords: Software Engineering, Specifications, Cost, Standard, Implementation

1 Introduction

In this paper, we describe a philosophical approach in realizing a software project's core requirements, namely , it is often possible to satisfy these requirements with greatly reduced time and cost by utilizing existing infrastructure to satisfy the requirements, rather than implementing every requirement.

In a traditional approach, we start a software project by gathering a core set of its requirements. Then decisions are made on the platforms, programming languages, and other infrastructure needed to develop a new system to realize the core requirements. Many books and papers have been written to discuss different approaches to manage every step of a software development project [1, 2, 3]. It is widely assumed that "all projects are bound by the Triple Constraints of Project Management: time, cost, and scope"—that if one factor changes, the other two must change just to maintain the quality to be the same, as show in Figure 1 [6].

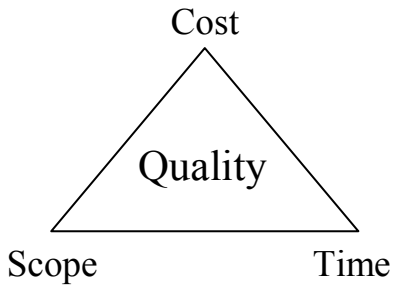


Figure 1 -- Triple Contraints of Project Management

This pervasive belief in the bounding triangle governs many projects. However, based on the authors combined working experience of more than fifty years involving countless successful software development projects, we have observed that the software industry in general has the tendency of implementing most every core requirement from scratch. Because the software industry accepts that reusability should be promoted as a well known OOP principle, we may temper this behavior by employing some third-party components and web services.

The introductions of component-based software development and web services have fostered the practices of using paid third-party components or services to reduce the software development cycle [8]. We agree with the authors that using third-party components generally reduces development time and cost. However the saved time may be absorbed by added efforts of searching and evaluating components, architecting the overall system to utilize selected components, and adding additional unit tests, integration tests, and system tests to confirm the acceptable behavior of the components [8].

Taking a deeper look, we noticed that the discussions of reusability often focus on reusing an organization's own code. We attribute this tendency in part to an understandable protection of Intellectual property. But even so, we'd like to call out that the project team has

already decided to implement, possibly with the assistance of third-party components, the features to meet the requirements.

The approach to first decide to implement features, then decide how to implement them, is evident in systems using web services. For example, Liu and He suggested that the development team needs to "assess what existing web services can the team delegate before it starts its own development." This again is intended to reduce the development cost after the decision is already made to develop. We'd like to ask, it is really necessary to always implement every core requirements?

In this paper we argue that, especially for software systems intended for internal users, rather than implement each core requirement, we should first look into whether the required features are already fulfilled by existing systems. If so, we should concentrate on integrations and training rather than implementation. With this approach, we believe that the end result is a solution with less development cost, less training cost, less maintenance cost, and shorter time to completion, while satisfying the software project's core requirements. Referring to the Triple Constraints of Project Management, it is entirely possible to deliver a software solution with lower cost, in shorter time, with a full set of features while maintaining the solution's quality. We call this the shortest path to satisfy software projects' core requirements.

Essentially, we believe there are two very different philosophies in meeting the core requirements of a software solution. One starts by investigating how to use existing tools to solve the problems on hand, the other takes the approach that "we know what software we need to develop to solve the problem, so let's start thinking about developing the solution.". We believe that it is possible to develop high quality software, while minimizing software development cost and reducing software delivery time in order to gain competitive advantages, by identifying existing software products that already fulfill some of the core requirements. To illustrate our software development philosophy, we will start with three cases where we need to provide industrial software solutions to some problems on hand.

2 Three software solutions supporting our argument

In this section we show three cases where development cost and time to market are drastically reduced, all attributed following the philosophical approach we championed in this paper.

2.1 Case 1

Company A sold several highly sophisticated teleconference endpoints to one of its clients, Company B. To fully utilize these endpoints, Company B requested Company A to provide a solution so its employees could easily include these endpoints in their meeting schedules. This request was in the sales contract. Both parties understood that should additional funding became necessary to develop a software solution, Company B would be willing to foot the bill as long as the amount was reasonable.

We were fortunate to be enlisted by Company A to look into the issue. After a couple of weeks of studying Company B's existing software, we were convinced that, with some minimum assistance from their IT department, their existing calendaring and meeting scheduling tool could fulfill this requirement with a few clarifications in the contract language. In addition, we strongly believed that utilizing Company B's existing software to answer this requirement was the best solution. We proposed our solution with supporting documents and suggested Company B's business manager to verify our proposal with their IT department. After several rounds of communication, Company B gratefully accepted our proposal to be considered as meeting this condition in the contract.

2.2 Case 2

In the world of teleconferencing, automatically connecting two or more teleconference endpoints at a predefined time is one of the basic operations [9]. In our case, connecting the endpoints is already possible, operated by a concierge, through existing mechanisms managed by the providers of teleconferencing devices.

We were asked to develop a software solution that could satisfy the following core functional requirements, among others: (1) to provide a UI to specify when to connect what set of endpoints for how long, (2) to check the availabilities of these endpoints, (3) to reserve the endpoints and inform the user that her request is honored if all the endpoints may be committed to the requested time and duration, (4) to

allow the user to change her request, and (5) to call an existing API to automatically connect the endpoints at the scheduled time. We were also given a long list of non-functional requirements such as a secure channel, platform independence, and a worldwide accessible UI, which led to an obvious decision that the UI must be web-based.

Complications arise when these endpoints may reside in separate intranets that do not necessarily trust each other. Also, endpoints can and should only be automatically linked to each other during the scheduled time, and it is preferable that this scheduling process is automated for security and confidentiality considerations, so that no human intervention is involved.

We quickly realized that when dealing with endpoints that do not belong to the same organization we cannot allow arbitrary users to schedule a connection to, say, the conference room of another company's CEO, for obvious considerations of security, privacy, and confidentiality. In particular: (1) an endpoint ID, like its owner's phone number, should not be treated as public information; (2) an endpoint's time is a valuable resource too and should not be allocated to just any requester; and (3) knowing whether a company CEO's endpoint is free at a certain time can also have business value and should not be readily available to arbitrary users. This realization clarified for us that basically all a web-based system should do is to collect requests for scheduled connections over a set of endpoints in possibly different organizations. Realizing that collecting schedule connection requests is the essential functionality was a major milestone of our project. However, immediately we faced the following problems for such a web-based application.

First, the application needs to support authentication, authorization, and user management. Building a solid user management system is not easy. Making it pass the set of security requirements prescribed by our company IT alone would make this a multi-quarter project. Second, since the application is on the Internet, we would have to find a way to bring in the data it collected automatically to ensure prompt processing of users' requests. A possible approach is to use SSH tunnels, initialized from our intranet. Quickly, it become clear that the security concerns make it impossible for our IT group to allow the use of tunnels in their production servers because doing so means they have to allow a hole to be opened in our DMZ. Last but not least, the team was instructed to implement a solution with absolute minimum impact on the existing infrastructure and with minimal development resources. This in turn introduced one fundamental non-functional constraint on our project's costs and time-to-market and called for a simple and reliable solution to get the job done. Even simply opening a secure tunnel through our DMZ could not be considered as meeting this "minimum impact" requirement.

After evaluating many possible solutions, we realized that, since all enterprises using our tools support some form of email plus calendar and scheduling system and each endpoint is assigned with an email address, we can meet the requirements that relate to collecting data simply through existing email systems. That is, we do not have to develop any new software for users to interact with! Figure 2 shows an example of enter all the necessary information using Microsoft Outlook. All we have to implement is the backend tool that processes emails that contain their requests. When a user needs to schedule a meeting, she does so as if she is scheduling any other meetings with one exception: she adds as invitees the assigned email addresses for all the endpoints she wishes to connect, plus a special email address to trigger our backend tools to process her request.

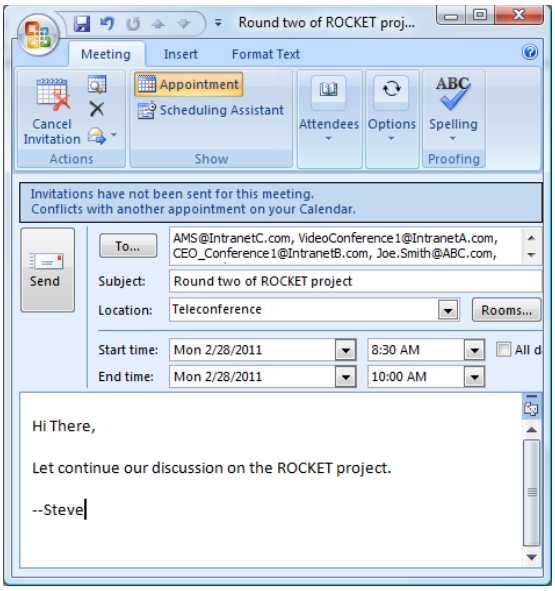


Figure 2 -- A sample meeting invitation including all meeting information and processing note's address.

The functions of the backend tool are straightforward and took our team a couple person-months to implement and test. All the information collection UI, which counts for majority of our system's core requirements, is implemented by the client's calendar and scheduling tool, such as Microsoft's Outlook

scheduling client, and the input to the backend tool is always an iCalendar record.

2.3 Case 3

During the development of a test system for product development, we tackled a requirement for a fully-generalized means for our users to define tests they wished to perform. Rather than design a new paradigm with its attendant complexities of development and training, we seized upon scripting as way for users to express their testing intentions. Scripting allows users to enter a sequence of functions to be executed, along with their parameters. To help our users succeed, we designed an extensible scripting language with a vocabulary suitable to a testing environment, such as simple simultaneous control over multiple test units. Thus we leveraged skills we could assume our testing community possessed—most everyone has been exposed to Basic programming, for example—while narrowing the application of those skills to keep our solution as simple as possible [10]. In this fashion we matched our solution to what existed already, namely a community and its vocabulary around testing principles and processes.

We didn't stop there. Recognizing that some testing needs go beyond the set of functions we provided in our scripting language, we further provided functions to execute more sophisticated test scripts defined in full-featured programming languages including Perl and Python. We still made simple the application of these languages by providing to them the common test "environment" shared by the external language and our own scripting language. This "back door" to extensibility further leveraged the scripting paradigm, enabled power users to perform arbitrarily complex tasks, and kept simple the use of our solution for most users. We also thereby tapped another existing solution space, namely the set of scripting languages power users employ to extend tools, and the community of scripting experts ready to use them.

Our success with this approach become evident over its thirteen years (and counting!) of application to a wide variety of specific test situations, expanding well beyond the original scope of testing targeted by our original solution. Prior to our solution, test operators were subjected to an endless series of unique project-specific test solutions, each requiring training and each featuring its own quirks. Because our solution leverages existing community practices and knowledge, it has fostered a long-term continuity of application. Truly it has fulfilled our early vision, inspired by former Apple Chief Evangelist Guy Kawasaki, to "let a thousand flowers bloom."

3 Discussion and recommended approaches

We've presented three cases of developing software solutions to solve problems on hand. All three projects were completed, however, with tremendous cost reduction and very high sustainability. They all are excellent representations of our "shortest path to satisfy software projects' core requirements" philosophy. We believe that a project manager responsibility should be more than just assuming that his most important priority is to build a software solution so the client's requirements are met regardless of the cost. Worse yet, even in the situation where building a new software system is the best solution, the project manager should not take the opportunity to add features and capability the clients have not requested even when the client can afford the cost and is willing to pay.

3.1 Identify systems that can answer new business needs

Let's consider the following extreme example. You were helping a client working on a business plan to open a pizza store with delivery services in, say, Seattle area. For vehicles used to carry out the delivery services, you could select either BMW 300 series or Ford Focus. We are certain that there will be cases where selecting BMW makes the best business sense. However, if the reason of selecting BMW is that it can reach 60 miles per hour from idle faster than the Ford Focus, then it is completely wrong. If we propose to design a special vehicle, with no special features related to pizza delivery, for this delivery service, then that would be a worse recommendation than selecting BMW because it can accelerate faster. The selection of vehicles would be more obvious if the owner already has several Ford Focuses.

From this example we can easily see that case 1 was an extremely successful project for both companies. Company B already has a fleet of "vehicles"—their email and calendaring system --that is capable of providing the required service, namely reserving some special resources. For reasons not known, they have not fully realized their "vehicles" capabilities provide the sought-after service. By pointing this out to our clients (Company A directly and Company B indirectly),

Company B avoided spending a large sum of money to have a piece of software built so the exact requirements in the contract were met, perhaps only to discover later that a much better solution fulfilling the core requirements already exists with their existing tool. In addition, Company B's IT department avoided having to support yet another piece of software. Most importantly, Company B was successful in utilizing the full potential of these teleconference endpoints as soon as their installations were completed. At the same time, Company A avoided developing, maintaining, and possibly constantly upgrading a piece of software that was not in their core business areas, and thereby was able to quickly refocus their engineering resources to more urgent and immediate issues. In addition, Company A was able to close this particular contract quickly.

One may argue that Company A lost a profitable business opportunity. We disagree, unless Company A wants to take the opportunity to expand its business segment. For a short term monetary gain, Company A may realize an insignificant amount of revenue without counting the extra efforts from accounting, legal, and management (all the way to VP). In the long run, Company A consistently needs to manage this piece of software. Considering all these, the solution in case 1 is really a win-win situation. For all the software solutions, this is as good as a "software solution" gets.

3.2 If we have to build, consider extending an existing system

When have to build something new, we should first consider if we could identify a few candidates to expend their functionality. When the new set of requirements is proposed by the current users of existing software, this approach is obvious. When the requirements are proposed by a different group of users, once again, we should not rush into development and be locked into that mindset. Rather, we should examine the possibility of focusing on selected few existing software solutions that can be customized. The benefits of doing can be many, mostly related to time and costs. We could reduce development costs by continuing use of the same infrastructure, and possibly even the same development team. Additional savings come from user training. Since the users are already familiar with the existing system, they can quickly learn the newly added functions and become productive with little or no training. Still more savings may come from code reuse. For example, generally speaking, the authentication and authorization can be extended to cover the new functions. Finally, the added maintenance cost on expanding a system is almost always lower than that of adding a new system.

Case 3 is such an example. By extending the capability of existing scripting language, the users can naturally broaden their usage of the tool to cover a new area. Clearly, developing a new system would most likely cost more, take longer to release for problem solving, and require more effort to train users on how to operate the tool.

3.3 Consider SaaS

When we are absolutely certain that we cannot utilize any existing software to fulfill the requirements, once again, before deciding to build, research the possibility of securing the service of a SaaS vendor. We recently evaluated a small company's internally built CRM system to look into ways to improve its usability and stability. Our recommendation was to scrap the existing system and subscribe to salesforce.com. The owner's comment was that he should have known us three years ago!

3.4 Beware the enthusiastic contractor

Contractors and third-party partners may find their easiest path to success through invention, motivated by self-interest and profit. Their enthusiasm and bias toward original creation can be infectious, and sometimes may deflect our course toward meeting core requirements. As we've illustrated here, we must resist—or at least fully understand the implications of—this natural tendency of our contracted colleagues. (And truth be told, we will find and must understand this same bias in our ranks. As Dr. McCoy famously observed in "Star Trek: The Motion Picture", "You know engineers, they love to change things.")

3.5 Hold the guiding vision dear, but adjust the means to achieve it

Throughout our discussion we've described the many ways a project may end up on a longer path to completion. We've alos described some approaches to keep to the shortest path. Perhaps the most fundamental of these is to first understand the guiding vision of our efforts, and to implant that vision firmly within those who will help realize it. Then we may explore freely and evaluate more objectively the various attractive means we've discussed here to achieve the vision by the shortest path.

4 Conclusions

In the old days when companies did not have a large number of software system, a decision of building a software solution, instead of buying, generally translated to building one based on some requirements. Nowadays, most companies have a large number of software solutions in operation. When a request to build a new piece of software comes, we should evaluate the requirements carefully to compose with a set of core requirements. Then we may find the shortest path—the approach that allows the organization to spend the least amount of money and be able to benefit in the shortest time, to realize the requirement.

When considering the overall cost of a software solution, the development cost is only part of it, covering only the first part of the solution's life-cycle. The costs of a complete project can come from training, enhancement, and maintenance. If users have to enter the same data multiple times, we need to add the cost of reconciliation as well. Because of this, we recommend that when looking for software solutions, we should not rush to developing new tools. Rather, we should evaluate whether we can utilize as much as possible of existing solutions first. Then we may consider a SaaS vendor before finalizing the decision to build.

Sometimes a single-vendor solution won't fit our needs entirely. Here the lessons learned from the open source community may help, where small extensions to existing solutions, combined perhaps with a "mash-up" of multiple solutions, can create the best result on the shortest path.

It is very difficult to quantify the savings as the result of shortest path to satisfy the core requirements. We would like to look into a few of our recent projects to see if we may use some of the earlier estimates to provide some guidance there.

5 References

[1] DSB, "Report of the Defense Science Board Task Force on Military Software", Office of the Under Secretary of Defense for Acquisition, US Department of Defense, September 1987

[2] I. Sommerville, "Software Engineering", 8 Ed, Addison-Wesley, 2006

[3] F. Brooks Jr, "No silver bullet: Essence and accidents of software engineering," IEEE Computer, pages 10--19, April 1987

[4] G. Forte, "Web Services Introduction", Intel Internal White Paper, February 2002

[5] J. Liu. and J. He, "Web-Based Software Development for Today and Tomorrow", in the Proceedings of "The 2002 International Conference of Internet Computing," June 24-27, 2002, Las Vegas

[6] J. Phillips, "Project management Professional Study Guide", 2 Ed, McGraw-Hill, 2006

[7] J. Liu and F. Liu, "Factors contribute to high costs of software projects", the 2009 International Conference on Software Engineering Research and Practice, July 13-16, 2009, Las Vegas, USA

[8] I. Crnkovic, S. Larsson, and M. Chaudron, "Component-based Development Process and Component Lifecycle," Journal of Computing and Information Technology, Vol. 13, No. 4. (2005), pp. 321-327

[9] A.Diane, InformationWeek. "Executives Demand Communications Arsenal." September 30, 2010. Retrieved October 5, 2010

[10] Ted W. Beers, "Fractal Frameworks: Finding Sub-Themes and Implementing Sub-Frameworks", IEEE International Conference on Information Reuse and Integration, Las Vegas, 2003

AGILE Burndown Chart deviation - Predictive Analysis to Improve Iteration Planning

A. Mr. Dhruba Jyoti Chaudhuri[1], B. Ms. Aditi Chaudhuri[2]
[1]Process Excellence Group, Tata Consultancy Services (TCS) Ltd., India
[2]Process Excellence Group, Tata Consultancy Services (TCS) Ltd., India

Abstract - *While AGILE development aims mainly at incremental development and delivering product (solution) in a time-boxed fashion; measurement framework is still not matured to assess and benchmark performance at organization and/or industry level. Burn down chart is one such key metric that tracks adherence to scope, effort and indirectly to schedule and is mandatory for daily Stand-up meetings. Though ideal target would be ZERO deviation; in real life we observe both +ve/-ve slippage(s). This paper is aimed at providing pointers, possible roadmap to analyze Burn down chart deviation to setup a predictable band or operating limit that would help improving Iteration planning to include suitable risk contingency reserve (+ve slippage: means possible push back some scope to Product Backlog, while –ve slippage: means probable provisioning of more scope into Iteration). This would ensure, greater assurance of completing planned Iteration Backlog within specific Iteration; the critical success factor of Agile planning/ execution.*

Keywords: "AGILE", "Burndown", "Iteration", "Planning", "Statistical" "Predictive"

1 Introduction

AGILE development primarily aims at developing and delivering product and/or solution in a time-boxed fashion, focusing on iterative and incremental development towards delivering tangible outcome (set of functionality/ system behavior). Each time-box is essentially an Iteration (also called as 'Sprint' or 'Scrum'), where a subset (called Iteration or Sprint Backlog) of total work scope (called as Product Backlog) is selected. This selected scope could ideally be completed in a specific Iteration and be ready to go live after Iteration is complete or with selected Release or through additionally planned System Test and/or Acceptance Test Cycle. During Iteration, a Task plan is prepared (for decomposed scope, as selected in Iteration backlog), by the team with Ideal (effort) hours is assigned to decomposed tasks/ activities. As completion of selected scope is extremely important, an Iteration Burn down Chart is prepared to show how total planned/ allocated effort would be consumed (Total to Zero) from start to end of Iteration. This planned Effort Burn Down is prepared keeping in mind, how much equivalent scope would be completed/ remained on daily basis. During Iteration execution, a revised effort estimate is put to account for remaining Iteration scope at the end of each day.

Though ideal (expected) Burn down performance would be a ZERO deviation; in real life it is often observed to be deviating ('actual' VS 'planned') both in +ve and –ve direction. As Iterations are time-boxed, to understand this behavior is of extreme importance; as +ve slippage indicates growing possibility of some amount of scope may have to be pushed back to Product or Release Backlog from Iteration backlog (due to probable unfinished scope), while –ve slippage means; possible under-utilized resource and more scope could have been provisioned into this iteration. It also guides to evaluate process effectiveness and trigger improvement cycle in regard to Scope management, Planning and Estimation, Risk Management, Issue (Impediments) Management and Resource Utilization etc.

This paper provides a framework and roadmap towards how, at an organization level, an expected and predictable band could be established and benchmarked, by analyzing a number of AGILE projects and a good number of Burndown chart behaviors. This requires periodic refinement and calibration based on future Organization data and Industry Benchmark, if available. This could be initiated at project level, then at organization level etc.

2 Burn down chart – what is it?

A Burndown chart is a simple but powerful tool to measure AGILE Project progress and manage deviation. Iteration Burndown represent, daily, the remaining work (basis iteration backlog) over specific iteration lifetime. It could be at Iteration, Release and/or Project level. It's a great management tool as it provides both project team and all other stakeholders with a common view of iteration and/or project progress.

Sample Burndown charts are shown below with possible interpretation and opportunities for improvement, towards better planning, monitoring and control of AGILE projects.

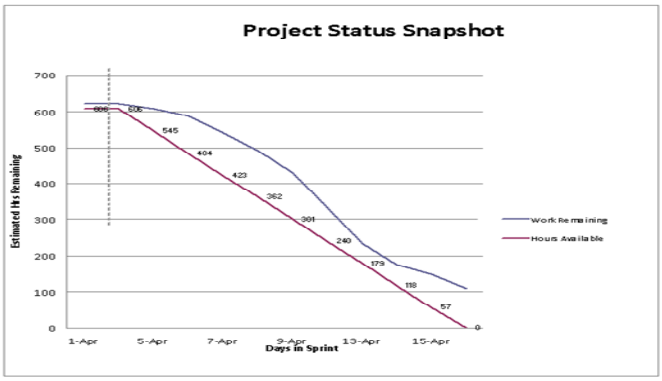


Figure 1: Sample Burndown chart #1

Possible interpretation: Iteration Planning and Estimation – Scope/ Task was under estimated.

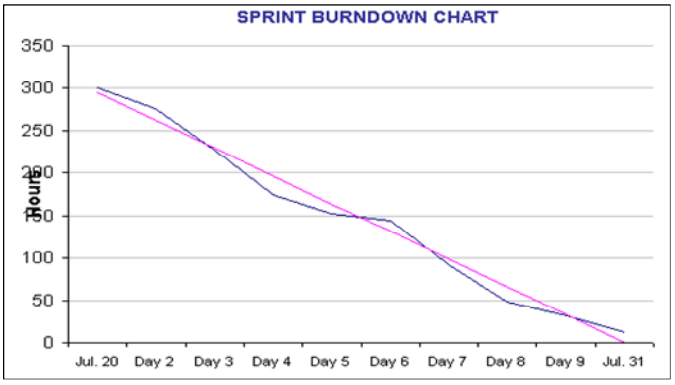


Figure 2: Sample Burndown chart #2

Possible interpretation: Normal expected performance and variation.

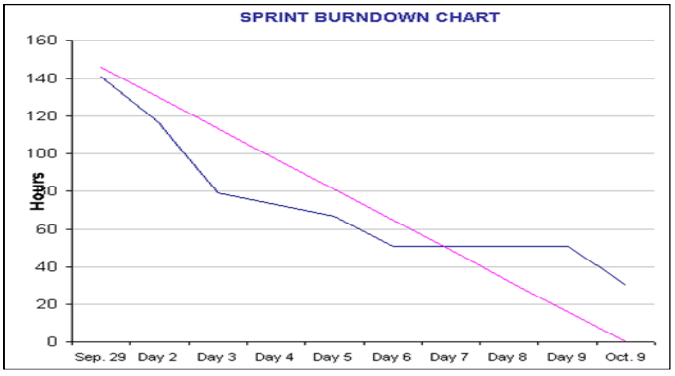


Figure 3: Sample Burndown chart #3

Possible interpretation: i) 1st part - Iteration Planning and Estimation – Scope/ Task may have overestimated and ii) 2nd part - Some tasks may have underestimated or some special cause/ impediments (issues) may have caused this.

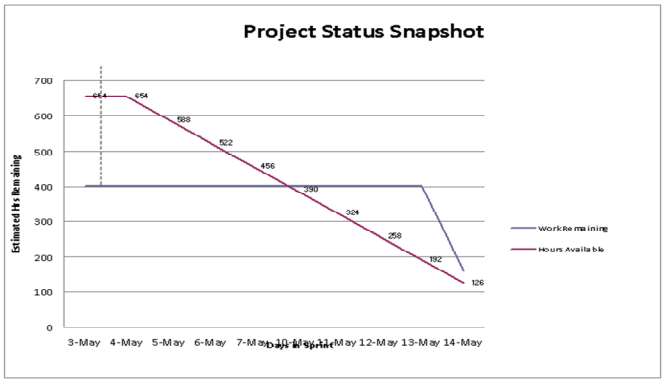


Figure 4: Sample Burndown chart #4

Possible interpretation: Typical scenario or special cause – for half of the iteration timeline, iteration Burndown didn't happen; then scope may have cut down (though scope adjustment is not allowed during iteration, and unfinished scope would automatically be returned to Product or Release backlog) to match rest of the iteration timeline. Or, the estimation was grossly on the higher side.

3 Improvement Opportunity, Objective and Business Case

3.1 Opportunity for Improvement

No Industry and/or Organization level guideline and Benchmark available to i) quantitatively understand and analyze Burndown Chart behavior, as a metric and ii) build predictability and enable better Iteration planning & execution.

3.2 Objective

Establish Organization, Unit, Account, Project Level Metric for i) Internal Benchmarking and ii) Drive process improvement and maturity in AGILE execution.

3.3 Business Case

Better planning of Iteration Backlog and Ideal Estimation to i) include risk reserve (both +/- deviations), ii) understand impediments and causes for these deviations and integrate with process improvement to attain next level maturity. iii) Improved Iteration Task Planning and Estimation.

4 Burn down chart analysis – proposed benefits

Table 1 – Benefit Articulation of Burn down chart analysis

Measurement	Helps understand current performance and ability to deliver
Project Planning	Helps in Iteration Planning; deciding on Iteration Backlog Scope better keeping in mind the operating/ predictable limit of deviation: planning adequate reserve for possible +ve slippage and having a backup scope or other tasks that could be additionally completed, in case of –ve slippage
Trigger for Causal; enabling improvement and optimization	To understand reasons for +ve and/or –ve slippage and correct process controls; for example; i) Story point or Value scoring guideline ii) Ideal Task estimation iii) Decomposition of User Story into Tasks iv) Coverage of various tasks like SDLC, Review/ Testing, Project Management etc v) Competence and Productivity of Team v) Planned VS expected Velocity vi) Dependency, Issues/ Impediments causing delay and resolution cycle time etc
Sprint Retrospective (Learning and Feedback loop)	Iteration Retrospection to analyze these deviation, identify improvement opportunities and adopt continuous improvement cycle
Baseline and Benchmark	Baseline performance in order to improve own performance and benchmark with other (different relationship, Organization Unit, Organization, Industry etc) performance and if better, adopt Best Practice(s) and/or learn from failure reasons
Effective Risk Management	Looking for key triggers and effective planning of Risk mitigation and contingency reserve
Time to Market	Greater assurance of completion of Iteration and Release Backlog on Time-boxed fashion

5 Burn down chart performance – predictive analysis approach

- Data collected for three different project execution, for same account and customer
- For each project, Burndown chart data captured, for various iterations
- Burndown chart behavior analyzed –
  - To understand daily deviations (planned vs. actual)
- Extreme outliers (special cause) eliminated
- Statistical Data Analysis done on data –
  - Normality & Descriptive Statistics, Box Plot, Dot Plot, Time Series, CNTL Charts etc
- Predictable Band (sample) selected based on outputs, it's interpretations and finally team's decision
- Same is carried out at Account/ customer level to understand overall predictability
- Need periodic calibration based on
  - Causal analysis, elimination of Special cause
  - Influence of Common cause
  - Corrective action taken to improve Iteration Planning and monitoring

6 Burn down chart monitoring – Data collection mechanism

Normally, day wise Ideal (planned/ expected) hours and actual (revised) hours to complete remaining scope of work, is captured; from which % variation is derived, as experienced and recorded in different days in iteration. Similarly, data is captured for other iterations, as well.

Day #	Ideal	Actual	% Deviation
1	280	285	1.79
2	245	250	2.04
3	210	200	-4.76
4	180	165	-8.33
5	145	149	2.76
6	120	120	0.00
7	80	70	-12.50
8	60	50	-16.67
9	20	25	25.00

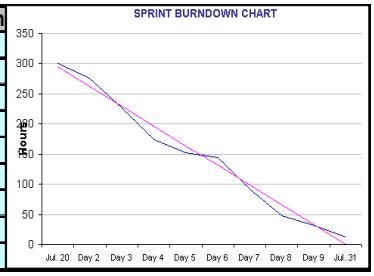


Figure 5: Sample Burndown chart and Data Collection

7 Burndown Chart Analysis: Determination of Predictive Band

7.1 Project level

Burndown chart analysis, for three sample projects (for same customer) were conducted using various standard techniques like Box Plot, Time Series, Dot Plot, Descriptive Statistics etc., and the same has been depicted below for reference. Three sample projects are represented as 'Case1', 'Case2' and 'Case3'.

Case 1

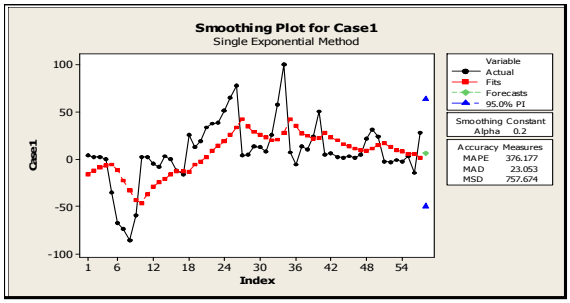


Figure 6: Time Series (Smoothing Plot) for Case 1

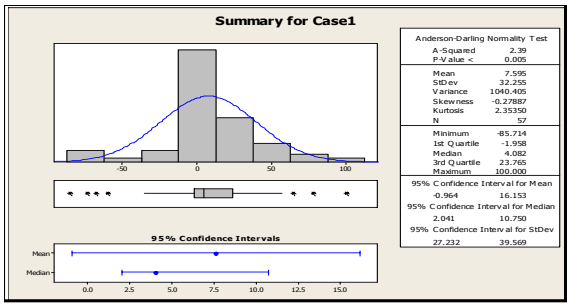


Figure 7: Normality & Descriptive Plot for Case 1

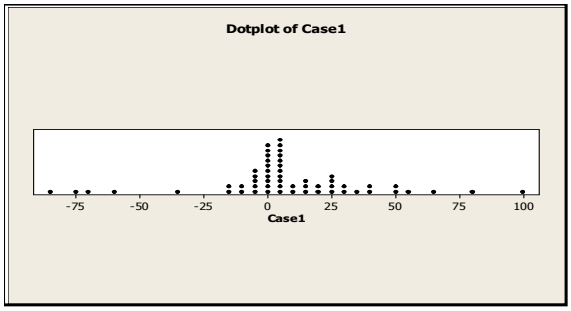


Figure 8: Dot Plot for Case 1

Case 2

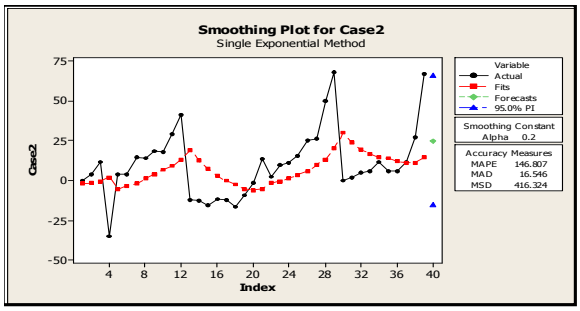


Figure 9: Time Series (Smoothing Plot) for Case 1

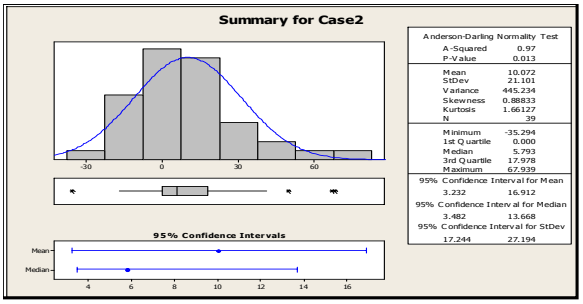


Figure 10: Normality & Descriptive Plot for Case 1

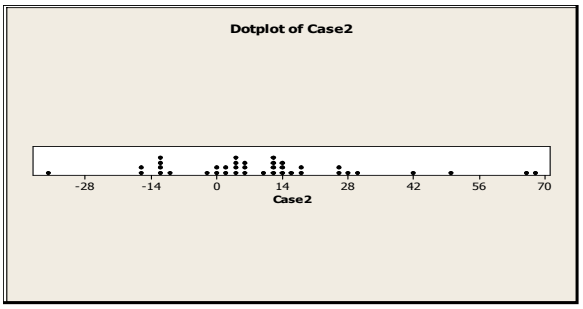


Figure 11: Dot Plot for Case 1

Case 3

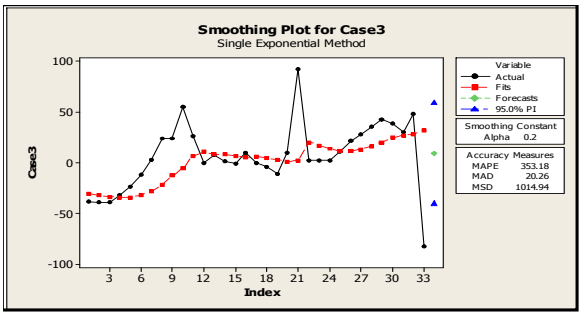


Figure 12: Time Series (Smoothing Plot) for Case 1

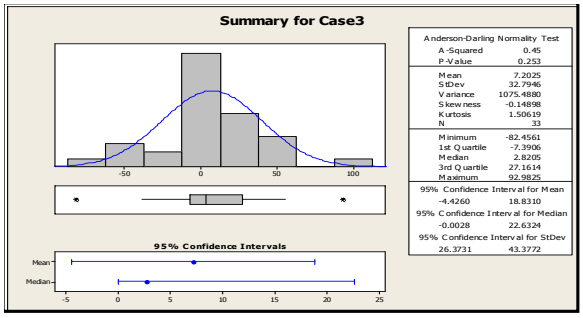


Figure 13: Normality & Descriptive Plot for Case 1

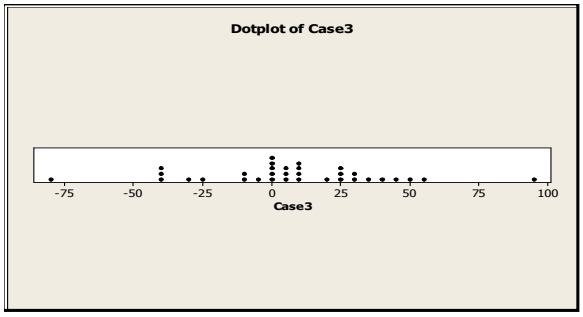


Figure 14: Dot Plot for Case 1

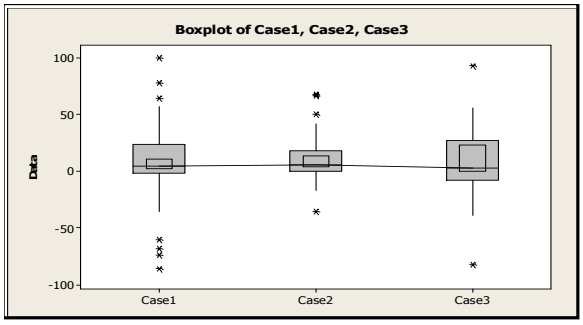


Figure 15: Box Plot for Case 1, Case 2 and Case 3

High level observations:

1. Case 1 and Case 2: Distribution is not Normal. Guidance is taken from other statistical analysis also.

2. Time Series (Smoothing): LCL and UCL limits are too wide, because of variations in both positive and negative directions.

3. Box Plots provide a good starting point to understand behavioral pattern and setup initial baseline (operating limits).

4. Time Series predicted value lies between Box Plot 95% CI for Median (except for Case 2, where variation is minimum)

5. Two prediction bands have been chosen as; i) ***Planning band***: This deviation could be considered as normal (most probable) scenario during Iteration planning, while ii) ***Risk band***: Risk/ Contingency reserve (Scope, Effort etc) may be required during Iteration planning.

6. For the *Planning band*; Box Plot 95% CI for Median has been chosen and for *Risk band* 1st Quartile and 3rd Quartile range has been picked up. Only exception considered for Case 2, as Time series prediction was out of both 95% CI for Median and 1st Q & 3rd Q range.

	Box Plot - 95% CI for Median			Time Series			Descriptive Stats	
Instance	LCL	UCL	Median	Predicted	LCL	UCL	1st Q	3rd Q
Case 1	2.04	10.75	4.08	6.24	-50.23	62.72	-1.95	23.76
Case 2	3.48	13.67	5.79	24.75	-15.78	65.29	0	17.97
Case 3	-0.002	22.63	2.82	9.28	-40.36	58.92	-7.39	27.16

	Predicted Band - Planning		Predicted Band - Risk	
Instance	LCL	UCL	LCL	UCL
Case 1	2.04	10.75	-1.95	23.76
Case 2	3.48	13.67	-15.78	24.75
Case 3	0	22.63	-7.39	27.16

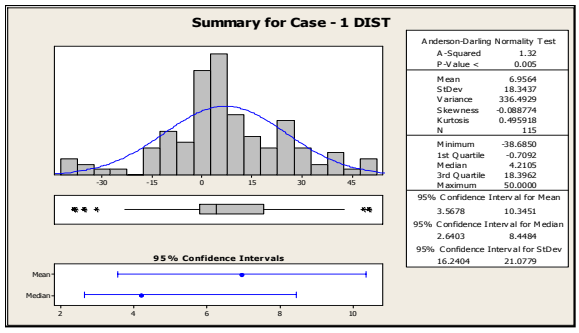
Figure 16: Sample selection of predictive bands

Note: In this case, the *'Planning Band'* has been considered as Voice of Process and is expected to be factored into normal course of Iteration Planning exercise, while for *'Risk Band'*, it would be advisable to plan for additional (contingency) reserve; as it is assumed that process is not matured enough and may still have wide variations in these range(s).

7.2 Account or Customer level

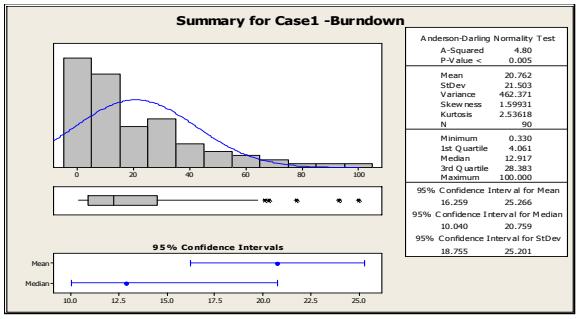
Here, all three (3) projects' data was analyzed together and consolidated, for a specific customer; similar process applied on sample outputs (Box Plot, Time Series prediction, Dot Plots and Descriptive Statistics) to come up with desired prediction band.

Probability distribution analyzed together with all deviation points (positive and negative) and also all positive and all negative separately to understand distribution patterns. As distribution found not normal, guidance is taken from other statistical analysis.



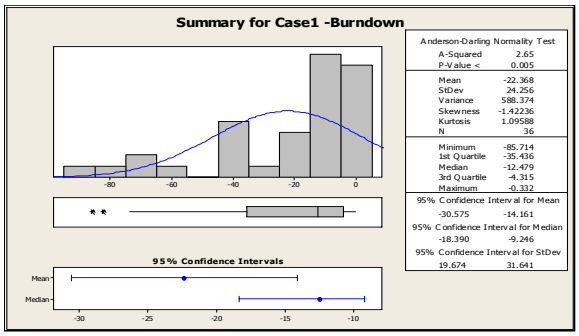
Overall Dist	Mean	Median	ST DEV		
	6.95	4.21	18.34		
95% CI	Median	LCL	UCL	1st Q	3rd Q
	4.21	2.64	8.44	-0.7	18.4

Figure 17: Overall distribution (basic statistics)



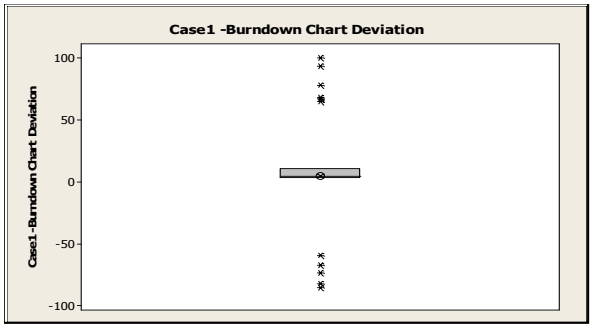
Positive Deviations	Mean	Median	ST DEV
	20.76	12.92	21.5
95% CI	Median	LCL	UCL
	12.92	10.04	20.75

Figure 18: Distribution of +VE deviations (basic statistics)



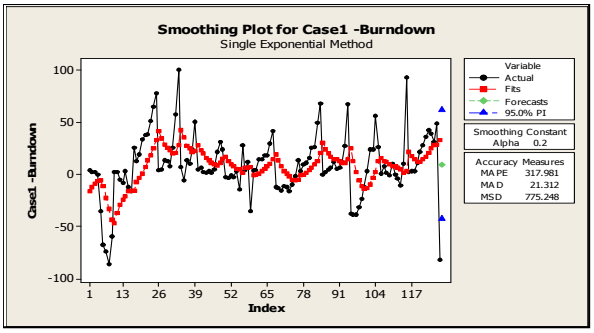
Negative Deviations	Mean	Median	ST DEV
	-22.37	-12.48	24.26
95% CI	Median	LCL	UCL
	-12.48	-9.25	-18.39

Figure 19: Distribution of -VE deviations (basic statistics)



95% CI	Median	LCL	UCL
	4.54	2.74	10.17

Figure 20: Box Plot statistics



95% CI	Predicted	LCL	UCL
	9.31	-42.81	61.52

Figure 21: Time Series (Smoothing) Plot

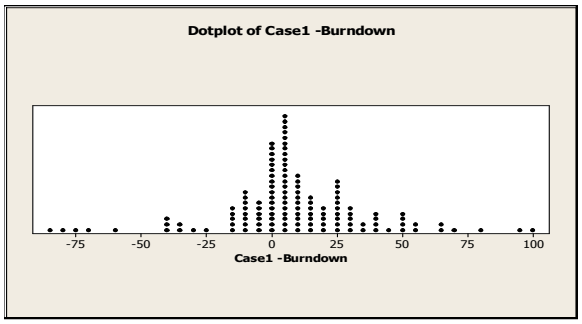


Figure 22: Dot Plot (Overall distribution)

Prediction Band	LCL	UCL
Optimistic Band	-9.25	10.17
Probable Band - Planning	-12.48	12.92
Possible Band - Risk	-18.39	20.75

Figure 23: Probable selection of three possible bands

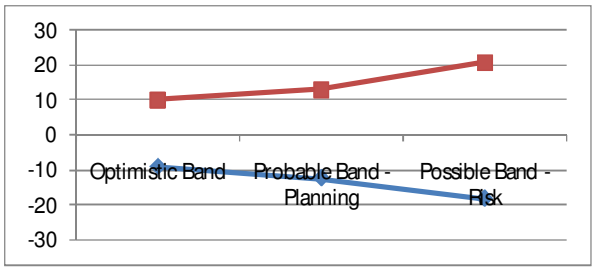


Figure 24: Graphical representation of possible bands

Notes:

i) Prediction bands, shown here, is indicative only and act as guidance, however, it may vary from project to project, customer to customer and team composition and maturity.

ii) Cells are highlighted with different colors to denote sample selection of possible prediction bands.

iii) Box plot is shown to be a good starting point taking guidance from Dot Plot to identify maximum density band.

iv) CNTL charts didn't provide useful prediction due to wide moving range variation, as deviation observed both in positive and negative direction, often.

v) Here, single exponential smoothing technique is used to observe meaningful prediction (if any), however, other variations of Time Series could also be used and accuracy measures (MAD, MAPE, MSE) could be observ

vi) Other predictive analysis techniques also could be explored for meaningful and best-fit outcome observed and found applicable.

vii) As we execute more Iteration(s), more Agile projects, calibration is required, as we gather more experience and data points.

8 Conclusions

This paper focuses on importance of analyzing Burndown chart behavior to come up with possible Prediction Band towards improving better planning and management of Iterations. Each deviation analysis helps us to identify improvement opportunities in scope/ task planning, estimation, competence, impediments, issue resolution, effort distribution, defect prevention, planning risk reserve (additional scope during iteration planning, if early finish or team sits idle etc).

9 References

[1] In-house tutorial on 'Statistical Techniques', Tata Consultancy Services Ltd., India
[2] Minitab Tool

SESSION

FORMAL SPECIFICATION, VERIFICATION, AND VALIDATION METHODS

Chair(s)

TBA

A Practical 4-coloring Method of Planar Graphs

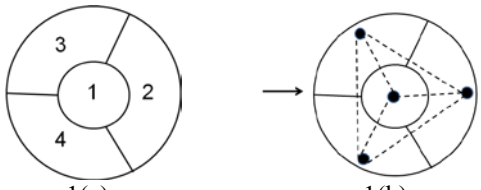
Mingshen Wu[1] and Weihu Hong[2]
[1]Department of Math, Stat, and Computer Science, University of Wisconsin-Stout, Menomonie, WI 54751
[2]Department of Mathematics, Clayton State University, Morrow, GA, 30260

ABSTRACT
The existence of a 4-coloring for every finite loopless planar graph (LPG) has been proven even though there are still some remaining discussions on the proof(s). How to design a proper 4-coloring for a LPG? The purpose of this note is to introduce a practical 4-coloring method that may provide a proper 4-coloring for a map or a LPG that users may be interested in. The coloring process may be programmable with an interactive user-interface to get a fast proper coloring.

Key words: 4-coloring, two-color subgraph, triangulation

1 Brief historical review

In 1852, Francis Guthrie, a college student, noticed that only four different colors were needed while he was trying to color the map of counties of England such that no two adjacent counties receive a same color. Since then, mathematicians had been working hard to prove this seemingly easy but full of traps problem. Clearly, four colors are definitely needed even for a simple map as shown by picture 1(a) below.



1(a) 1(b)

Picture 1: face coloring is equivalent to vertex coloring

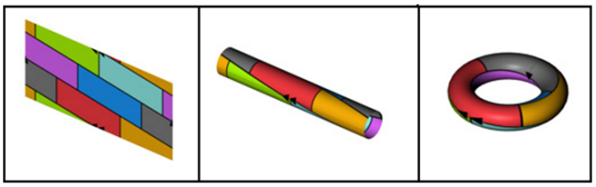
Coloring a map is known as face coloring problem. We denote a finite loopless (every edge joins two distinct vertices) planar (can be drawn on a plane without edge crossing) graph by LPG. The dual graph D(G) of a LPG G is defined as the graph that represents each face of G by a vertex, and two vertices are adjacent in D(G) if and only if the two faces share boarder in G, for instance, the dotted graph in 1(b) is the dual of 1(a). Clearly, D(G) is a LPG as well. A LPG is 4-colorable if its dual graph is vertex 4-colorable. So, face coloring is studied via vertex coloring.

Two American professors Kenneth Appel and Wolfgang Haken proved the 4-coloring theorem in 1976 using a computer [1],[2],[3]. However, many mathematicians have been working on a theoretical proof [4], some individuals tried to give a counter proof [5]. Some mathematicians have been studying the properties and invariants of the 4-coloring of LPGs as well [6].

An interesting coloring formula was conjectured by Heawood in 1890 [7]: For a given genus g > 0, the minimum number of colors necessary to color all graphs drawn on the surface of that genus is given by

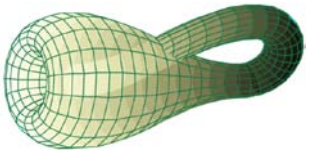
$$\gamma(g) = \left\lfloor \frac{7+\sqrt{1+48g}}{2} \right\rfloor .$$

For example, with genus $g = 1$, it is a doughnut-shaped object. $\gamma(1) = 7$ means the chromatic number is seven. Picture 2 shows that the seven regions (with seven different colors) can be topologically put on a torus such that the seven regions are mutually adjacent each other on the torus. So, it clearly requires at least seven colors to be able to give a proper coloring of maps on the torus.



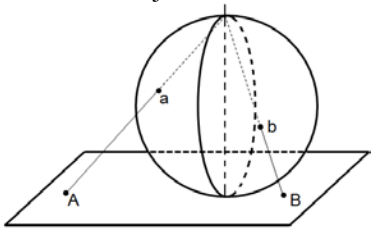
Picture 2: Seven coloring torus [9]

Ringel and Youngs [8] proved that Heawood conjecture is true except the Klein bottle which needs six colors only (picture 3).



Picture 3: the Klein bottle [9]

A plane map can be seen as a map on a globe (picture 4), and hence 4-colring of planar graphs is equivalent to 4-coloring of the maps on a globe. A torus with g=0 is isomorphic to a globe. So, even if the proof of 4-coloring theorem was the most difficult one, we may see the 4-coloring of maps is the lowest case of Heawood conjecture.



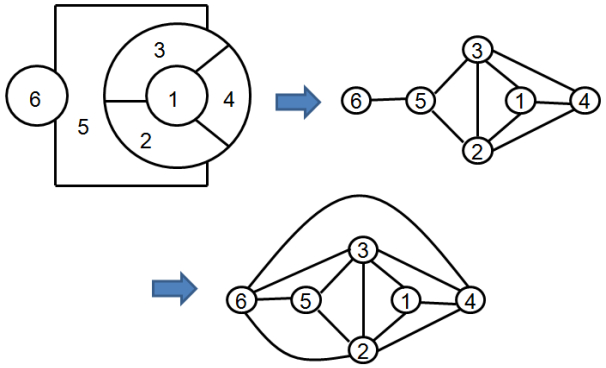
Picture 4: A globe to a plane

The 4-coloring problem is still drawing attention around the world. People also would like to see an effective method

that can provide a 4-coloring for a map or a LPG that people are interested in.

2 Some definitions related to a LPG

A triangulation of a LPG G may be obtained by adding edges, without edge crossing, to G until each face of G becomes a triangle (picture 5). Such a triangulation is also said to be a maximum planar graph. Obviously, a LPG is definitely 4-colorable if its triangulation is 4-colorable.



Picture 5: a map, its dual graph and triangulation

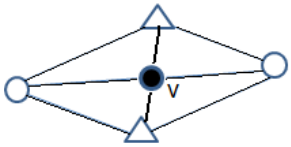
Mathematicians have studied the properties of such maximum planar graphs. The famous Euler formula says that, let p, e, and f be the number of vertices, edges, and faces of a planar graph G, respectively, then $p-e+f=2$. If G is also a triangulation, then one can derive the following properties from Euler formula:

(a) $e=3p-6$,

(b) $f=2p-4$, and

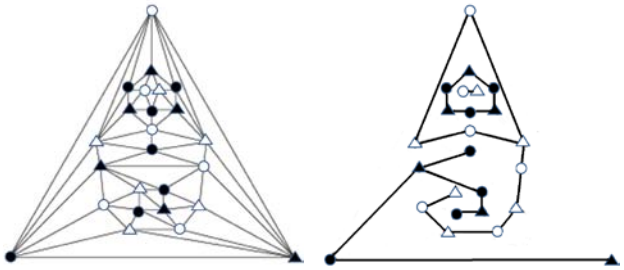
(c) $6p-2e=12$. [(c) shows that in a triangulation planar graph, there is at least a vertex with degree ≤ 5.]

Definitions: Suppose a planar graph G receives a 4-coloring using colors c_1, c_2, c_3, and c_4. Assume that vertex v is properly colored referring to its neighborhood. We say that vertex v is adjustable if the color of v can be assigned a different color, without changing any color of its neighbors, such that v itself is still properly colored referring to its neighbors. For example, the vertex v in picture 6 is adjustable. An edge is said to be a "bad-edge" of a coloring if its two endpoints received the same color. We will fix the improper coloring on a bad-edge via color switching schemes.



Picture 6: an adjustable vertex

Assume that G receives a proper 4-coloring. A two-color subgraph is a c_i-c_j subgraph $G_{i,j}$ spanned by all vertices of G that are colored by c_i or c_j. There are six two-color subgraphs. We consider them as three pairs of two-color subgraphs: $G_{1,2}$ and $G_{3,4}$, $G_{1,3}$ and $G_{2,4}$, and $G_{1,4}$ and $G_{2,3}$. For example, picture 6(a) shows a proper 4-coloring of the Heawood graph. Picture 6(b) shows a pair of two-color subgraphs.



6(a) 6(b)

Picture 6: the Heawood graph

In a two-color subgraph, the total degree of each component must be even, and hence the total degree of a two-color subgraph is even. These two-color subgraphs also have the following properties:

Each component of a two color subgraph is of cycle and/or tree structure, i.e., each component is a cycle, or a tree, or a "cycle-tree" (tree is hanging on the cycle). For example, picture 6(b) shows a cycle and a tree for the two black color symbols; a tree and a cycle-tree for the two white color symbols. Each $G_{i,j}$ has only even cycle(s), if any, for a proper 4-coloring. So, trying to break any two-color odd cycle of an improper 4-coloring is a critical scheme for getting a proper one.

A fact that can help on finding a proper 4-coloring is that if a vertex is of degree three or lower, we may remove it from the graph since the color of this vertex is either uniquely determined by its neighbors or it can be an adjustable vertex when we put them back. This action can be performed recursively before design a coloring.

We define a reference path P of a graph G to be a path of G such that no subset of vertices of P forms an odd cycle in G. A reference path can be properly colored by two colors that are alternately applied along the path.

3 The practical method of finding a 4-colring of a LPG

This note introduces a programmable process that may provide a proper 4-coloring for a general map or a LPG. We do not assert this process must be a success (that would claim we have a proof of the 4-coloring theorem), however, we have successfully found a proper 4-coloring for all examples we have had via this method.

Assume that we are given a planer graph G.

Step 1: Find and remove all vertices with degree three or lower recursively. [Stop if the remaining number of vertices is less than 5. A proper 4-coloring is obvious.]

Step 2: select a reference path P and color it by two colors, say c_1 and c_2, alternately. [User should be able to select the reference path via user-interface.]

Step 3: (Initial coloring)
Let S_k be the ordered set of all vertices v of G such that distance $D(v,P)=k$ in G. The order of vertices in S_k is created by a depth first search. For example, if x is the last vertex added to S_k, then a neighbor y of x with $D(y,P)=k$, if any, should be added to S_k next. So, S_k is ordered like a circle around the reference path P.

Color S_k by colors c_3 and c_4 if k is odd, or by c_1 and c_2 if k is even. This step gives an initial coloring for all vertices of G. [There may have bad-edge(s) that will occur if either any S_k itself forms an odd cycle or a subset of S_k forms an odd cycle in G.]

Step 4: (Correction step)
Detect any bad-edge(s). If no bad-edge, it is a perfect coloring and stop.
Otherwise, fix the bad-edges one at a time (of course, if lucky, more than one bad-edge might be fixed simultaneously). Select a bad-edge (u, v) and assume u and v received color c_1.

(4.1) For each two-color subgraph $G_{1,j}$, j=2, 3, or 4, detect whether there is a component that consists of only even cycle(s) and/or tree(s), and that contains exactly one of u and v. If so, the bad-edge may be fixed by swapping these two colors on this component, and return to the beginning of step 4. Otherwise, continue.

(4.2) For each two-color subgraph $G_{1,j}$, j=2, 3, or 4, detect whether there is a component that consists of two-color even cycle(s) and/or tree(s) containing edge (u, v). If so, recolor this component with the two colors, and return to the beginning of step 4. (u, v) should be fixed.
Otherwise, there is an odd cycle containing (u, v) in each two-color subgraph. (u, v) cannot be fixed now.

(4.3) This step does not fix any bad-edge, but change the color status. *Users should have a chance to determine a starting vertex to perform any one of the following sub-steps.*
(4.3.1) Detect whether there is a component C in a two-color subgraph with one color of S_k and one color of S_{k+1} such that C consists of even cycle(s) and/or tree(s) only. If such a component exists, exchanging the two colors along C, then return to the beginning of step 4.
If necessary, this detection can be performed several times using different starting vertex, and/or different pair of colors between S_k and S_{k+1}. Continue if nothing can be done.
(4.3.2) Detect whether there is a two-color cycle that contains edge (u, v) such that we can change the color c_1 that u and v both have now to another color. [This does not create or fix a bad-edge.] If yes, do so and return to the beginning of step 4. Otherwise, continue.

(4.3.3) Detect whether there is an adjustable vertex. If so, assign a different color to the adjustable vertex and return to the beginning of step 4. Otherwise, continue.

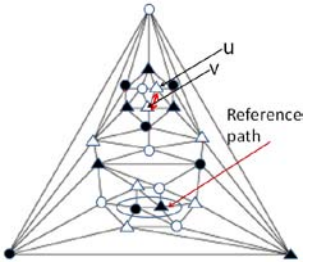
Step 5: Assign a new reference path P and go to step 3 to restart the coloring process. The reference path P can have one or more vertices. *User may select P by experience or observation.*

4 Examples

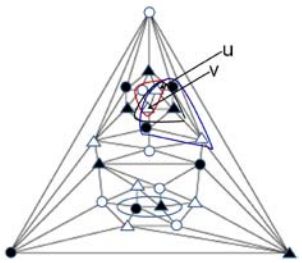
As you have seen above, we represent the four colors using the symbols ▲, ●, Δ, and O.

Example 1: The Heawood graph (picture 6(a)) (see [10]). Heawood presented this graph as a counterexample to the proof of 4-coloring theorem by Alfred Kempe.

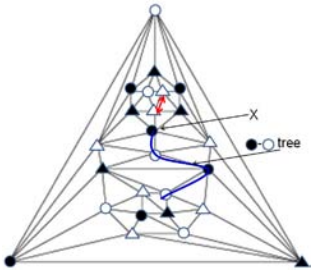
4-coloring process by the practical method:
Step 1: Nothing can be done.
Step 2: Select the circled two vertices as the reference path and color it using the two black color symbols. (7(a))
Step 3: Color the ordered set S_k alternately using two white color symbols if k is odd; two black color symbols if k is even. There is a bad-edge (u, v) with color Δ in S_5 (7(a)).
Step 4:
(4.1) No such component.
(4.2) There is an odd cycle within each of (Δ, O), (Δ,▲) and (Δ,●) subgraphs as shown on picture 7(b). This implies we have no way to fix (u, v) now.



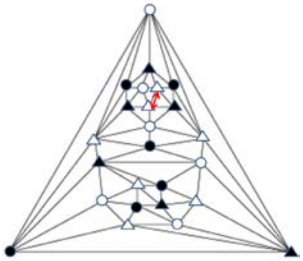
Picture 7(a)

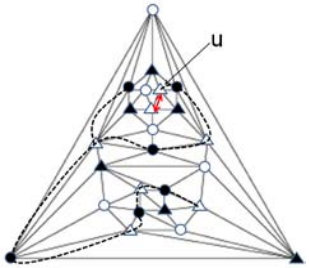


Picture 7(b)

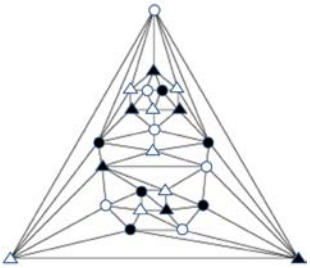


Picture 7(c)


Picture 7(d)



Picture 7(e)

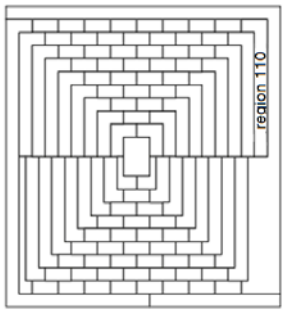

Picture 7(f)

Picture 7

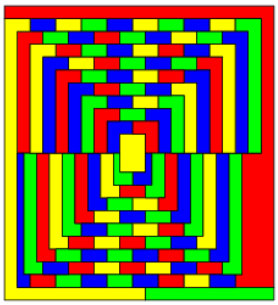
(4.3) starting with vertex x – picture 7(c)
(4.3.1) There is a ●-O tree cross S_k s from vertex x (7(c)). Swap color ● and O along this tree, and return to the beginning of step 4. (7(d))
(4.1) There is a Δ-● component that is a tree containing vertex u only (indicated by the broken-curve in picture 7(e)). Swap colors Δ and ● along this component and return to the beginning of step 4.
No more bad-edge. A perfect coloring obtained. (7(f))

Note: This coloring process is not unique. Referring to picture 7(d), v by itself is a trivial Δ-● path, so we can simply switch colorΔ to ● for v to get a perfect 4-coloring as well.

Example 2: Gardner's April Fools' Day puzzle
In 1975, Martin Gardner "claimed" the map of 110 regions (picture 8) requires five colors to get a proper coloring. Of course, Gardner just made a fun on the Fool's Day. A proper 4-coloring of Gardner's map had been found couple years after he published the map (picture 9).

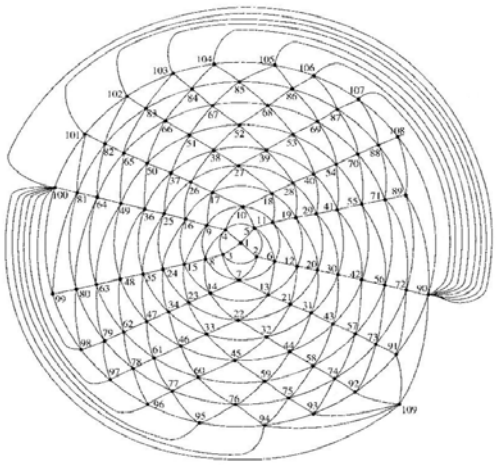


Picture 8

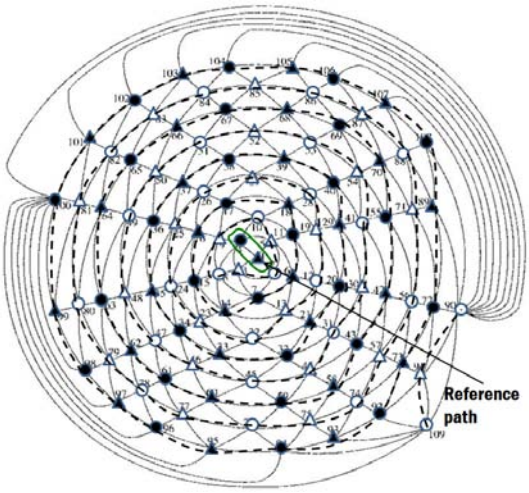

Picture 9

We illustrate the coloring process by find a proper 4-coloring of the Gardner's graph.

Since region 110 (see picture 8) is enclosed by three regions only, so its color will be uniquely determined by the surrounding three regions and hence we can remove it from the map. The dual triangulation of Gardner's map (with region 110 removed) is given by picture 10. We took this picture from Xu's book [6] and corrected a minor error.

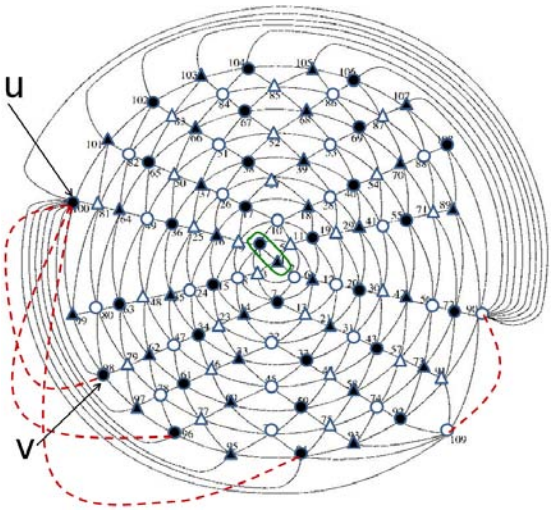

Picture 10 Gardner graph

Step 1: Done already.
Step 2: Select the two vertices at the center of the graph as the reference path. Color this path by ▲ and ●. (picture 11)
Step 3: Search for S_k, for k=1 to 9. Color each S_k using two white color symbols if k is odd; or two black color symbols if k is even. S_9 is of only three vertices. Unfortunately, there are odd cycles within S_8 and S_9. (picture 11)



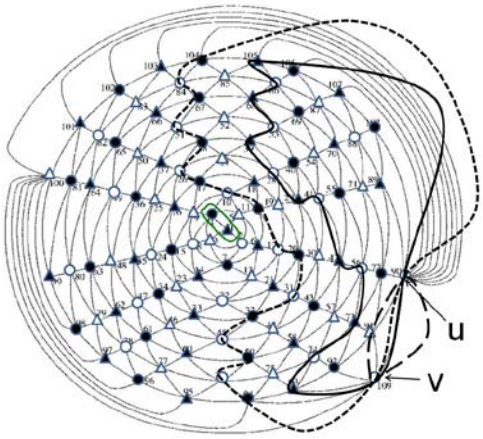
Picture 11

After coloring by distance to the reference path P, there are several bad-edges. See the broken lines in picture 12.
Step 4: Select a bad-edge (u, v) with color ● (picture 12).



Picture 12

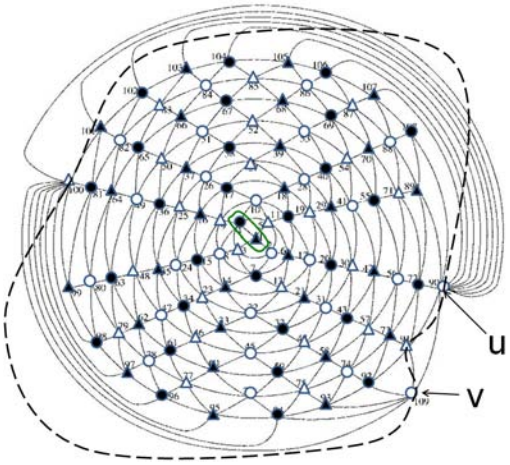
(4.1) Starting with vertex u there is a ●-Δ path that contains vertex u only. Swap ● and Δ and return to the beginning of step 4. [Luckily, three bad-edges on the left side are all fixed. Picture 13]



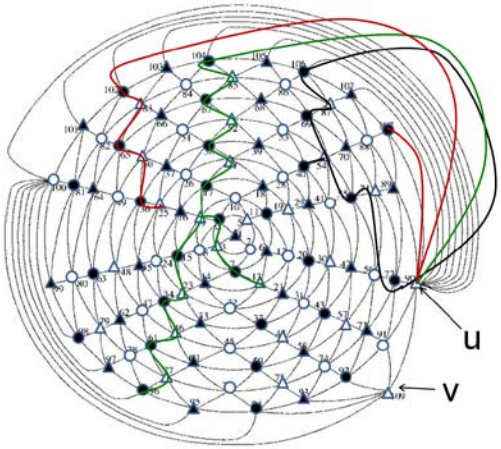
Picture 13

Step 4: The only bad-edge (u, v) is on the right side which received color O for both u and v (picture 13).
(4.1) No such component in anyone of two-color subgraphs.
(4.2) There is an odd-cycle found in each (O,▲), (O, ●), or (O, Δ) subgraphs. This implies that we have no way to fix the bad-edge by switching a pair of colors now. Picture 13 shows the three two-color odd cycles.

Under this situation our process goes to (4.3). Hope (4.3) can update the color situation and possible have a chance to get rid of the bad-edge.



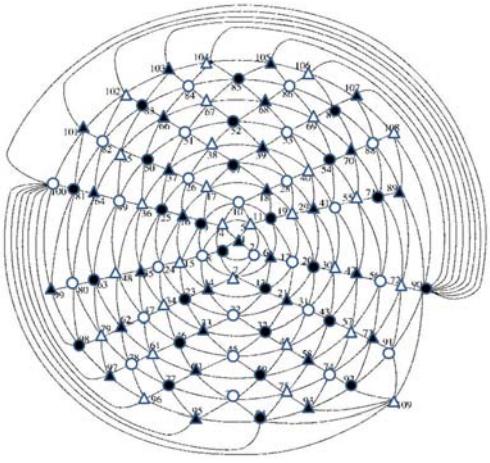
Picture 14



Picture 15

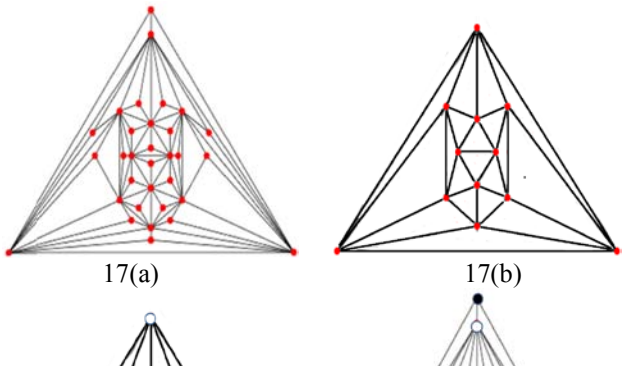
(4.3.2) There is an even Δ-O cycle found (picture 14). Exchange color Δ and O along this cycle and return to the beginning of step 4. [Now, the color of both u and v is (picture 15).]

(4.1) Starting with vertex u, there is a Δ-● component that consists of even cycles and trees and that does not contain vertex v (picture 15).
Swapping colors Δ and ● for this component results in a perfect coloring of Gardner's graph as shown in picture 16.



Picture 16 A perfect 4-coloring

Example 3:
Let's study the Triakis Icosahedral graph [11] (picture 17(a)). To directly design a proper 4-coloring for this graph is not trivial since there are many K_4s. However, removing all vertices with degree three (step 1 of our coloring procedure) gave graph 17(b). We omit the details since it is not difficult at all to assign a proper 4-coloring for 17(b) as shown in 17(c). A proper 4-coloring obtained by adding those removed vertices back with a proper color (see 17(d)).



17(a) 17(b)

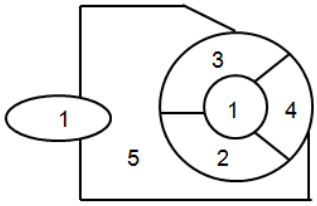
17(c) 17(d)

Picture 17

5 Conclusions

The 4-coloring process we introduced here can be done manually. It is an interesting exercise that can be a practical project for students. This process is also programmable. In fact, the fundamental detection process includes DFS or BFS search, creating the ordered sets, assigning colors, and swapping colors actions. User-interface is an important factor for getting a fast result.

We finish this note by indicating that an actual map might requires more than four colors if there are some countries whose territory has more than one region and the territory of each country must be of the same color. For example, in the following map, if the two regions both labeled 1 must be of the same color, it would enforce to use the fifth color.



6 REFERENCES

[1] K. Appel and W. Haken, Every planar map is four colorable. Part I. Discharging, Illinois J. Math. 21 (1977), 429{490. MR 58:27598d

[2] K. Appel, W. Haken, and J. Koch, Every planar map is four colorable. Part II. Reducibility, Illinois J. Math 21 (1977), 491{567. MR 58:27598d

[3] K. Appel and W. Haken, Every planar map is four colorable, A.M.S. Contemporary Math. 98 (1989). MR 91m:05079

[4] Electronic Research Announcements of the American Mathematical Society, . Volume 2, Number 1, August 1996

[5] http://www.superliminal.com/4color/4color.htm

[6] Shouchun Xu, 图说四色问题 (Picturizing 4-coloring problem), Peking University Press, ISBN 978-7-301-12800-8, 2009.

[7] Heawood, Map colour theorem, Quart. J. Pure Appl. Math. 24 (1890) 332-338.

[8] Ringel and Youngs, Solution of the Heawood map-coloring problem, Proc. Nat. Acad. Sci. USA 60 (1968) 438-445.

[9] Wikipedia, http://en.wikipedia.org/wiki/,
File: Projection_color_torus.png

[10] Wolfram MathWorkd,
http://mathworld.wolfram.com/HeawoodFour-ColorGraph.html

[11] Wolfram MathWorkd, http://athworld.wolfram.com /TriakisIcosahedralGraph.html

A mathematical model for prediction of the human performance based on the personal features

Konstantina Georgieva, Reiner R.Dumke, Anja Fiegler

Dept. of Computer Science, University of Magdeburg, P.O. Box 4120,
D-39016 Magdeburg, Germany
T-Systems, Services & Solutions, Magdeburg, Germany

Abstract – *Design of Experiments is a well-known method used to build mathematical models that describe complicated processes. We decided to use this method in order to describe the personal performance process in connection with the individual features in the software development cycle. Using the FMEA over the different roles in the software process to obtain the most important human factors and then applying another well accepted psychological theory - the Big Five we visualize the factors that influence the human productivity and then use them in order to build our experiment.*

1 Introduction

Human is the driving wheel of our world. Human skills, ideas and imagination are the inspirations for all surrounding inventions and technologies, cultural and intellectual progress. The trace of human touch and sense is in each emerging technology, theory, business solution and machine. Humans' touch is in almost every object in our world and we have to realize that when there is a human act – there might be a human error too.

Human error examples could be everywhere: small problems at home; design problems in a usability form; machine construction and usage; people to people and human to machine interaction. Consequences are also numerous from small quarrels to catastrophic life-threatening events. Because of this, looking from the point of view of the software development it is very important to understand the human factors in it and to try to evaluate them for their criticality.

In our paper we try to investigate the different features in the human character that influence the individual's performance, to find the most important of them and then to predict the human performance based on these characteristics.

In the literature until the moment there have been made a lot of attempts to say which are the most important human features that influence the software process but because of the complexity of the human being they haven't been listed with the needed accuracy. We will take a look first at the already published materials over human factors, human errors and human malfunctions and then we will summarize the human characteristics that we consider for crucial and we will try to evaluate them.

Reason [1] defines human error as a planned sequence of activities to achieve some intended outcome. He differs between mistakes and slips and defines slips as unintended actions and mistakes as intended actions, which don't lead to the expected goal.

These slips and mistakes can have different reasons and can be grouped according to the classification of Rasmussen [2] into: skill-based; rule-based and knowledge-based. Skill-based performance is explained with automatic and unconscious actions; rule-based is associated with following procedures and knowledge-based is conscious problem solving. Rasmussen proposes also factors that influence the human behavior: social and management climate, type of the overworked information, emotional condition, physiological stressors and physical workload.

Yingxu Wang [3] proposes taxonomy of human factors in software engineering and builds a behavioral model of human errors, which is expressed in evaluation of the performed task. This model is based on the fact which actions are conducted and which not in the process of performing a certain task.

In their work Hillson and Webster [4] speak about the connection between emotions and risk behavior and try to show the relation between emotional literacy and work attitude.

In the Human Reliability and Error in Transportation Systems [5] the important factors affecting the productivity of the individual work are said to be the Stressors, for example: *Poor training or skill; Complex task; Poor work layout; etc.* Although that the book is for Transportation Systems all these factors exist also in the software development process.

Dayer [6] summarizes the factors that influence the human reliability in two groups: internal and external factors. Internal are trust and working climate and external are family, health and the Maslow's pyramid of needs [7].

One of the most recent papers [8] describes the human factors as: Personal competency; Experience and leadership; Team performance; Availability of skilled personnel; Commitment; Personnel loyalty; and different specific Working skills.
Yanyan and Renzuo [9] explain the Psychological background of human behavior as a mixture of human knowledge, emotion and intention. They try to explain the relationship between software engineering and knowledge, and in the same time to include the human factors that influence this knowledge.

These scientific works take different views over the human factors in the software process and although that try to connect the human behavior with the human mistakes, no one has tried to observe the personal characteristics and the resulting from them working performance. Under personal characteristics we understand the individual traits that are important for every employee and that influence the working process and the conduction of mistakes. Based on this recognized gap we have summarized the most important human features that affect the work quality and we have built a model that predicts the individual performance.

2 Adopting FMEA and Big Five for the software process

In order to reach the first step in our approach – the discovery of the critical human features in the software development process, we have started with looking for the most common software company structure. Then we have summarized the main roles in it and have described their responsibilities. An organizational structure is the "formal system of task and reporting relationships that controls, coordinates, and motivates employees so that they cooperate to achieve an organization's goals" [10]. There are three basic types of organizational structure [11]: Functional Organization – it is structured according to the business functions and the staff members are grouped by specialty; Project Organization - all activities are projects and the people are in project teams and Matrix Organization - this is a combination of the other two that implies their advantages.
In software development organizations - a typical organization is the matrix one. For all the roles inside we have described the responsibilities and then we have applied the well-known FMEA method in order to find the weak places in each role where possible problems and failures can occur and the possible human features that influence the concrete process.

The Failure Mode and Effect Analysis (FMEA) is a well known method for failure analysis in the manufacturing industries, introduced in the late 1940's for military usage by the US army and later on found big usage in the aerospace and rocket development. With the years it becomes more and more popular and is used in different big industrial companies. Stamatis [12] proposed the use of FMEA in the information systems. He claimed that computer industry failures may result from software development process problems, coding, systems analysis, systems integration, software errors, and typing errors and that all these failures can origin from the work of the people in the concrete development process.
In order to complete effective FMEA one must follow a constructive approach. The method brakes down the process into basic sub-processes and examines potential problems in each of them. The FMEA can be generally summarized into the following steps: Discover failure modes; Recognize the causes for them; Assign a RPN (Risk Priority Number) and Proposing mitigation strategies. We have selected this approach because it gives the possibility of consequential analysis of all process-types and it also results with a RPN which helps in prioritizing the critical factors.
After applying the FMEA method to the different roles and their responsibilities, we have gained tables with all expected problems for each role [13] with the corresponding human factors that stay behind.
Processing these tables and analyzing all different human characteristics connected with the different failure modes we have ended with 74 different personal factors that influence the working process. Having the advantage of the FMEA method we had a RPN (Risk Priority Number) for each of them and this gave us the possibility to decide which the critical factors are. Here is the list with the selected factors, which we will evaluate on the next step.

- attention
- competence
- cooperation
- hardworking
- knowledge
- mental overload
- talkativeness
- satisfaction
- communication
- concentration
- coordination
- intelligence
- management
- personal organization
- understanding

We are making the evaluation with the help of the 'Big Five' theory and few additional questions.

The Big Five [14], [15], [16] was originally described in the 1970's and it states that the most human personality traits can be assigned to five broad dimensions of personality, regardless of language or culture. These are Extraversion, Agreeableness, Conscientiousness, Emotional Stability and Imagination/Intellect. Each of the factors has six facets [17] which describe its complex structure. The Big Five is now the most widely accepted and used model of personality and because of this we are taking it as a milestone in our research.

We have referred the above mentioned critical factors to the Big Five personality traits in order to evaluate them. As we were not able to connect all factors to the Big Five features we had to add separate questions for estimation of the experience, productiveness and motivation. As these are additional factors that originally don't belong to the Big Five model, we have decided to take the personal estimation of the participants and this of their supervisors and managers. In this way we were

able to build a value for these subjective features, based on the own evaluation and on that of the supervising personnel.

Having this done we have ended with 63 questions (50 from the Big Five and 13 additional) [18, Zoho] which we have given to people working in 5 software companies on different positions with different experience and at different age so that we have ended with 55 usefully fulfilled questionnaires that we are using in order to conduct our statistical analysis on the next step.

The matching between the different factors summarized from us and the Big Five looks like this:

- Extraversion: communication, talkativeness
- Agreeableness: cooperation, understanding
- Conscientiousness: attention, hardworking
- Emotional Stability: concentration, coordination, management/personal organization
- Imagination/Intellect: intelligence

Here come the two new factors added from us Experience and Motivation that are also very important for a complete picture over the working process.

- Experience: competence, knowledge
- Motivation: mental overload, satisfaction

3 Design of Experiment applied over the personal factors

Processing the 55 complete test results and summarizing the data, we have asked ourselves how we can evaluate the influence of these seven factors over the software development process. After a long literature research we have chosen to use the Design of Experiment method because of its advantages [19]:

- gains maximum information from a minimum number of experiments;
- studies effects individually by varying all operating parameters simultaneously;
- takes account of variability in experiments or processes themselves;
- characterizes acceptable ranges of key and critical process parameters contributing to identification of a design space, which assurances quality.

The Design of Experiment [19], [20] gives the possibility to model a process without knowing how exactly the input factors influence the end product. Because of this it is the best to be applied in our situation.

Conducting all the steps in the design of experiment, the first fundamental point was to select the number of the input factors and the type of the experiment. We have performed a correlation analysis over the data gained with the questionnaires to see the connection between the Performance and every from the personal factors. Here are our results from the correlation analysis:

Correlation (Motivation, Performance) = 0.96
Correlation (Conscientiousness, Performance) = 0.72
Correlation (Intelligence, Performance) = 0.59
Correlation (Agreeableness, Performance) = 0.41
Correlation (Experience, Performance) = 0.25
Correlation (Extraversion, Performance) = 0.19
Correlation (Emotional Stability, Performance) = 0.128

Led by these results and the knowledge that correlation values from 0.3 have medium and from 0.5 have big importance [21], it was easy to decide that we will consider the first four factors.

Observing that the motivation has a value of 0.96 which means almost linear dependence with the performance we have decided to build a full-factorial experiment from the type: 2^3, considering the rest three factors: conscientiousness, intelligence and agreeableness by fixed values for the motivation. We have chosen also to use the central composite rotatable plan, because of its advantages [22] [23]:

- Ensures the invariance of the plan and of the parameter of optimization by rotating the coordinate system around its centre;
- The model obtained by the rotatable plan describes the response surface with equal accuracy in all directions of the coordinate axes;

The experiment is conducted according to the following table 1, where the X values are actually the input factors in coded form and Y is the output factor. After conducting the experiment we will receive a mathematical model from the type:

$$\hat{y} = b_0 + \sum_{i=1}^{k} b_i x_i + \sum_{\substack{i=1 \\ i<j}}^{k} b_{ij} x_i x_j + \sum_{\substack{i=1 \\ i<j<g}}^{k} b_{ijg} x_i x_j x_g + \ldots + \sum_{i=1}^{k} b_{ii} x_i^{\,2} + \ldots$$

And for this model we have to make a check if all the coefficients *b* are correct and if the model is adequate. This is done with the corresponding statistical formulas, one from which is the Fisher Criterion: $\hat{F} = \frac{s_{adeq}^{\,2}}{s^2[y]}$

Table 1.Matrix for rotatable plan of second level - type 2^3 (factors are in coded form)

	Number of the experiment	X_1	X_2	X_3	y
Full factorial experiment 2^3	1	-1	-1	-1	y_1
	2	+1	-1	-1	y_2
	3	-1	+1	-1	y_3
	4	+1	+1	-1	y_4
	5	-1	-1	+1	y_5
	6	+1	-1	+1	y_6
	7	-1	+1	+1	y_7
	8	+1	+1	+1	y_8
''Star'' points	9	-1,682	0	0	y_9
	10	+1,682	0	0	y_{10}
	11	0	-1,682	0	y_{11}
	12	0	+1,682	0	y_{12}
	13	0	0	-1,682	y_{13}
	14	0	0	+1,682	y_{14}
Experiments in the canter of the plan	15	0	0	0	y_{15}
	16	0	0	0	y_{16}
	17	0	0	0	y_{17}
	18	0	0	0	y_{18}
	19	0	0	0	y_{19}
	20	0	0	0	y_{20}

We will not give any more details about Design of Experiments because of the lack of space, for the interested reader, the method is good explained in the following literature : [23], [20], [24], [25].

We have conducted the complete algorithm of Design of Experiments for the three selected factors by fixed values for the motivation. As the motivation factor showed almost a linear connection with the performance, we were able to conduct three experiments in three fixed values of it – 85%; 70% and 55%. Our data analysis showed that the changes of the other factors are significant by these three different values of the motivation factor.

After conducting the three experiments and validating them with the corresponding statistical formulas we have gained three mathematical models that describe the connection between the input factors and the output factor – the performance.

These models are identical and because of this we will show here only one of them by motivation of 55 % as it is considered for the most critical one, because as lower the motivation as lower the performance is.

The model looks like this:

Performance [%] = pr(co, int, agr) = -523.021607 + 3.139297*co + 7.793311*int + 4.525325*agr – 0.002677*co*int + 0.001674*co*agr – 0.021694*co*co – 0.046799*int*int – 0.031346*agr*agr ,
where co=conscientiousness; int=intelligence and agr=agreeableness.
Having this equation we can predict the performance of every employee based only on his/her psychological features (motivation, conscientiousness, intelligence and agreeableness).

We can also see the response surface of the performance, displayed with Matlab on the next figure. We can build different graphics for every two factors, for example here we have intelligence and conscientiousness and the other two factors have to be fixed (motivation=55% and agreeableness=55%), otherwise we cannot display the graphic on a 3D figure.

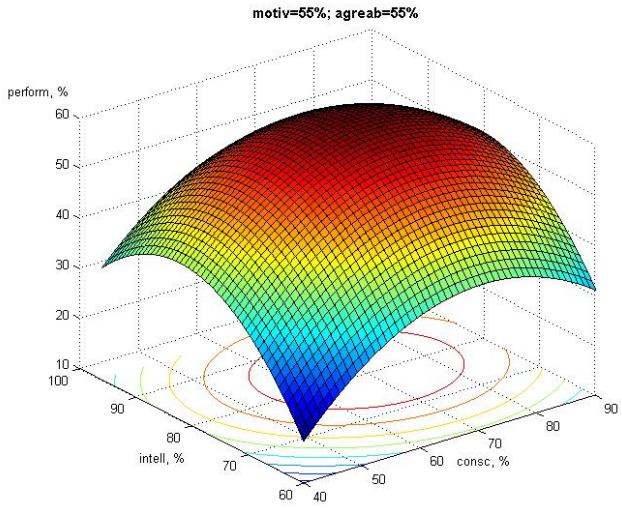


Figure 1.Response surface by (motivation=55% and agreeableness=55%)

When observing the figure we see that we have a maximum and according to our calculations this maximum point of the performance is 71.4% and is achieved by co=70.2%, int=81.2% and agr=74% by fixed motivation of 55%. This means that when we have people with the following values we will gain maximum performance from them and when the values are different the resulted performance will be lower. This result gives us also one another observation that by motivation of 55% the maximum performance that we can achieve is 71.4% and not more. This is explained with the fact that our data showed the linear connection between these two features. The other two experiments by motivation of 70% and 85% end with very similar equations, only with different coefficients. Because of this we will only discuss the maximum reached performance by motivation of 85% and it is 94.5%, where the values of the other factors are: co=70.6%, int=81.4%, agr=74%.

We can clearly see that the difference between the values of the factors co, int and agr is so minimal that could be ignored, this gives the deduction that the motivation is the main factor that in combination with the other three influences the performance at most.

4 Conclusion

We have used well-known methods (FMEA, the Big Five theory and Design of Experiments) in order to conduct our study over the personal features and their influence on the working process and we have found out that the four most important factors for a productive employee are: Motivation, Conscientiousness, Intellect and Agreeableness.
.
We have built a mathematical model that gives us the possibility to predict the human performance based on these personal characteristics. This enables us to evaluate the people in the software development and to prognosticate their productiveness.

Using this new method for analysis we can distinguish between different candidates who are the most suitable one. Anyway there arises also the question - how can we influence these four factors, especially the Motivation, because we have seen that it is the main factor that brings performance in the software process.

In future we have to validate the model with additional observations over employees and we also have to find out how the stimulated motivation brings better performance and how to influence that.

5 References

[1] Reason. (1990). *Human Error.* New York: New York: Cambridge.
[2] Rasmussen, J. (1982). Human errors: A taxonomy for describing human malfunction in industrial installations. *Journal of Occupational Accidents* .
[3] Wang, Y. (2005). On cognitive properties of human factors in engineering. *In Proceedings of the Fourth IEEE international Conference on Cognitive informatics (July 31 - 31, 2005)* (S. 174-182). ICCI. IEEE Computer Society, Washington, DC.
[4] Hillson, D., & Webster, R. (2006). Managing Risk Attitude using Emotional Literacy. *PMI Global Congress EMEA Proceedings.* Madrid, Spain.
[5] Dhillon, B. (2007). Basic Human Reliability and Error Concepts. In *Human Reliability and Error in Transportation Systems* .
[6] Dayer. (2007). Consideration of Human Errors in Risk Management.
[7] Maslow, A. (1987). *Motivation and personality.* NY, USA: Harper Collins.
[8] Islam, S., & Dong, W. (2008). Human Factors in Software Security Risk Management. *ACM.*
[9] Yanyan, Z., & Renzuo, X. (2008). The Basic Research of Human Factor Analysis Based on Knowledge in Software Engineering. *International Conference on Computer Science and Software Engineering.* IEEE.
[10] Lamar, U. (2010). *Organizational Structure.* Abgerufen am 19. 02 2010 von http://dept.lamar.edu/industrial/Underdown/org_mana/Org_Structure_George.htm
[11] Kurbel, K. (2008). *The Making of Information Systems Software Engineering and Management in a Globalized World.* Springer Berlin-Heidelberg.
[12] Stamatis, D. H. (1995). *Failure mode and effectanalysis: FMEA from theory to execution.* Milwaukee: WI: ASQC Quality Press.
[13] Georgieva, K., Neumann, R., & Dumke, R. (2010). Psychological-based Measurement of Personnel Performance. *Proceedings of the 2010 International Conference on Software Engineering Research & Practice, WORLDCOMP 2010 (SERP 2010)* (S. 543-546). Las Vegas Nevada: CSREA Press.
[14] Tupes, E., & Cristal, R. (1961). *Recurrent Personality Factors Based on Trait Ratings.* Technical Report ASD-TR-61-97, Lackland Air Force Base, TX: Personnel Laboratory, Air Force Systems Command.
[15] Digman, J. (1990). Personality structure: Emergence of the five-factor model. *Annual Review of Psychology 41* , S. 417-440.
[16] Goldberg, L. R. (1993). The structure of phenotypic personality traits. *American Psychologist , 1* (48), 26–34.
[17] Joh, O. P., Robins, R. W., & Pervin, L. A. (2008). *Handbook of personality: theory and research.* Guilford Press.
[18] Georgieva, K. (kein Datum). Abgerufen am 1. March 2011 von Zoho: https://challenge.zoho.com/dellly
[19] Shivhare, M., & McCreath, G. (June 2010). Practical Considerations for DoE Implementation in Quality By Design . *BioProcess International , 8* (6), 22–30 .
[20] Montgomery, D. C. (2008). *Design and Analysis of Experiments.* John Wiley & Sons Inc.
[21] Cohen, J. (1988). *Statistical power analysis for the behavioral sciences - 2-nd edition* (Bd. 112). Routledge Academic.
[22] Khuri, A., & Cornell, J. (1996). *Response Surfaces:designs and analysis.* Marcel Dekker.
[23] Myers, R. H., Montgomery, D. C., & Cook, C. (2009). *Response Surface Methodology Process and Product Optimization Using Designed Experiments.* Wiley, New York.
[24] Box, G. E., Hunter, W. G., & Hunter, J. S. (1978). *Statistics for Experimenters,An Introduction to Design, Data Analysis, and Model Building.* Wiley, New York.
[25] Mason, R. L., Gunst, R. F., & Hess, J. L. (2003). *Statistical Design and Analysis of Experiments with Applications to Engineering and Science.* John Wiley & Sons.

A Proof-Based Approach to Detect Vulnerabilities in C programs

Amel Mammar[1], **Liu Penfei**[2]
[1]Institut Telecom SudParis, CNRS/SAMOVAR, Paris, France
amel.mammar@it-sudparis.eu
[2]INRIA, Phoenix / Bordeaux-Sud, Talence, France
Pengfei.Liu@inria.fr

Abstract— *This paper presents a formal approach to detect vulnerabilities in a C program using the B formal method. Vulnerabilities denote faults that may be introduced unintentionally into programs making them behave incorrectly. Such faults (or programing errors) may lead to unpredictable behavior and even worse well-motivated attackers may exploit them later to cause real damage. Basically, the proposed approach consists in translating the vulnerable aspects of a C program into a B specification. On this B specification proof and model checking activities are performed in order to detect the presence or absence of vulnerabilities. Compared to the existing vulnerability detection techniques, a proof-based approach permits to eliminate false alarms and denial of service attacks.*

Keywords: Security; Vulnerability detection; Mapping rules; B formal method; Proofs; Model Checking.

1. Introduction

Software vulnerabilities are programming mistakes or bugs that compilers fail to detect. Buffer and arithmetic overflows are well-known examples of vulnerabilities. With the fast proliferation of complex, open and distributed systems, such software vulnerabilities become a real issue that needs to be fixed since these systems are often used in critical domains like transportation, finance, politics, etc; which makes attackers more and more motivated to cause important damage by exploiting these security breaches [6]. Above all, software defects are expensive, according to a 2002 National Institute of Standards and Technology study, software errors cost the U.S. economy an estimated 59.5 USD billion annually [15].

To detect software vulnerabilities, several techniques have been developed [12], [11]. Roughly speaking, these techniques can be classified into two classes: *static* and *dynamic* analysis. Each one has its own particular strengths, however one technique is not sufficient to deal with all the possible errors. In general, several techniques must be applied to detect a larger range of vulnerabilities. The main drawback of static techniques is the high number of false positives they can produce, whereas dynamic techniques suffer from denial of service since the detection is performed at program run-time. A more detailed description of these existing techniques is given in Section 7.

In this paper, we propose a formal approach, based on the B method, to detect vulnerabilities in a C program. Our approach consists in converting the C program into a B specification on which it is possible to perform correctness proof and model checking activities to detect programming bugs or errors. By converting a C program into a B specification, we generate the necessary and sufficient information about the C vulnerable statements. To deal with a buffer overflow for instance, we generate information about the size of the buffer and also the size of the data it holds at any time.

The rest of the paper is structured as follows. The vulnerabilities we consider in this paper are presented in Section 2. The B method is introduced in Section 3. Then, Section 4 gives an overview of the proposal which is described in detail in Section 5. The actual detection of vulnerabilities using proof and model checking activities is described in Section 6. Section 7 presents the related work and a comparison with our approach. Finally, we conclude and present some potential future work.

2. Software vulnerabilities

In [16], several types of vulnerabilities are described together with their countermeasures and how we deal with them using the B method. These vulnerabilities include buffer overflows and different arithmetic overflows and underflows but also format string vulnerabilities. For the sake of space, this paper only presents buffer overflows and arithmetic vulnerabilities.

Buffer overflows. To store the content of variables and buffers, a program reserves a specific amount of memory space. A buffer overflow occurs when a program attempts to put more data in a buffer than it can hold. A buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code. Buffer overflows are the favorite exploit for hackers.

Arithmetic vulnerabilities. These vulnerabilities occur when a calculation produces a value that is greater in magnitude than a given register or storage location can store or represent. Each integer type in C has a fixed minimum and maximum values that depend on the type's machine representation, whether the type is signed or unsigned, and the type's width (e.g., 16-bits vs. 32-bits). At a high level, integer vulnerabilities arise because the programmer does not take into account the maximum and minimum values. Over one hundred C integer vulnerabilities have been publicly identified to date, some of which have resulted in serious disasters such as rocket malfunction.

3. Overview of the B method

Introduced by J. R. Abrial [1], B is a formal method for safe project development. B specifications are organized into abstract machines. Each machine encapsulates state variables on which operations are expressed. The set of the possible states of the system are described using an invariant. The invariant is a predicate in a simplified version of the ZF-set theory, enriched with many relational operators. Operations are specified in the Generalized Substitution Language (GSL) which is similar to the DijkstraŠs guarded command notation. A substitution is like an assignment statement. It allows us to identify which variables are modified by the operation, while avoiding mentioning which ones are not. The generalization allows the definition of non-deterministic and preconditioned substitutions. Refinement is the process of transforming a specification into a less abstract one. A refinement can operate on an abstract machine or another refinement component. In B, we distinguish behavioral and data refinement. In this paper, we only use behavioral refinement that aims at eliminating non-determinism and coming close to the control structures used in the chosen target programming language. Behavioral refinement includes, for example, weakening of preconditions, the replacement of parallel substitution with a sequence one, etc. To ensure the correctness of a B specification, a set of proof obligations is generated for each B component. At the abstract level, these proofs aim at verifying that the invariant of the system is satisfied after the execution of each operation. Of course, such an invariant is assumed to be satisfied before an operation is executed. For each a given invariant Inv and an operation op whose precondition and substitution are P and S respectively, the following proof obligation is raised: $(Inv \wedge P) \Rightarrow [S]Inv$. Notation "$[S]Inv$" denotes the predicate obtained by applying the substitution S to the variables of Inv. Refinement proofs permit us to check the correctness of each concrete operation with respect to its abstraction. We assume that readers are familiar with B method and more details can be found in [1].

4. Overview of the approach to detect vulnerabilities

Our method to deal with vulnerabilities using the B method includes four phases (See Figure 1):

- In a first step, the C code is transformed into an abstract form. To this end, we have defined a sub-set of the C language that include the declarations of buffers and arithmetic data. More details can be found in [16].
- From the abstract representation of the C code, a B formal specification is automatically generated. The B specification contains information about the vulnerable elements of the code. In case of a buffer for instance, we consider its size and also the size of the data it holds at any time.
- To detect the presence/absence of vulnerabilities, proof obligations are generated using the proof obligations generator (POG) of AtelierB. The prover of AtelierB is then used to discharge them. The failure of the prover to automatically demonstrate a goal (the proof) can be due to two different reasons: either it lacks tactics and needs human help and intervention or the goal is false.
- To be sure that an unproved goal corresponds to a vulnerability, we use the model-checker ProB [22] in order to exhibit a state that satisfies the opposite (complement) of this goal. This means that the goal is really false.

The following section describes the generation of a B specification from a C program in order to detect vulnerabilities.

5. Generation of a B model from a C program

In this section, we describe how it is possible to use the B method to detect vulnerabilities in a C program. Our method is based on extracting information about the vulnerable functions or data that are used in the program.

5.1 The subset of ANSI C

Our approach considers a subset of the ANSI C language satisfying the following assumptions:

- the ANSI-C program has been already preprocessed, e.g., all the $\#define$ directives are expanded (inline expansion).

- the supported C types are primitive integer and array.

- function calls are expanded (inline expansion), the return statement is replaced by an assignment (if the function returns a value).

- all the arithmetic operations contain two operands, that is, they are of the form ($z := x\ op\ y$). Multi-operand operations are transformed into binary operations by introducing temporary variables. For instance, operation $(a = a_1 + a_2 + a_3)$ is replaced by the following two binary operations: $a_{temp} = a_1 + a_2$, $a = a_{temp} + a_2$. This

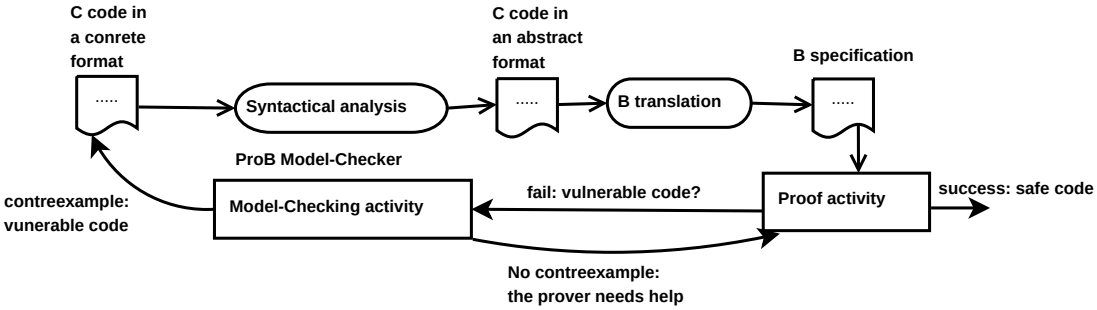


Fig. 1
DETECTING C VULNERABILITIES USING THE B METHOD

transformation is necessary in order to detect any arithmetic overflow. Indeed, multi-operand operations may mask some overflows like in statement $(x = maxint + 1 - 1)$ where the sub-assignment $(x = maxint + 1$ produces an overflow while its equivalent $(x = maxint)$ does not.

- the declaration of variables, buffers and files,

- the predefined C functions like input/output functions ($printf$, $scanf$, etc.) and also the functions on buffers ($strcpy$, $strncpy$) and files ($fscanf$, $fgets$),

- the assignment, the sequence and the choice (**IF**) statements.

A complete grammar of the subset of ANSI C supported by the proposed approach to detect vulnerabilities is provided in [16]. Hereafter, we describe how we obtain the B model from a C program written according to this subset.

5.2 The B model architecture

Recall that our objective is not to build an equivalent B model for a C program. Indeed, our goal is to build an abstraction of a C program by extracting the useful information to detect vulnerabilities that may be present in a C program. Figure 2 depicts the architecture of the B model that we derive from a C program. The B model we derive from a C program is composed of two B components:

- At the abstract level, a B machine is defined. It contains the variables of the C program and an invariant to state the types of these variables but also to express that the C program we would like to build should not contain any vulnerability, that is, all the variables are in their respective ranges. An operation $Main$ is specified to model the behavior part of the C program. This operation consists of a non-deterministic substitution that states that some values are assigned to variables according to the invariant. In other words, the operation represents the behavior of a C program that assigns values to its variables without producing any vulnerability.

- At the refinement level, a B component that refines the first one is created. The refined operation $Main$ of this component contains the translation of the instructions of the C program we are dealing with. By this way, the refinement proofs will demonstrate that the refinement component is conform to its abstract specification, that is, it does not contain any vulnerability.

Buffer declaration	$size\_buff$
char buff[N]	N
char buff[N]="*hello_world*"	N
char buff[]	1
char buff[] $= \{'a','b','c'\}$	3
*char *buff ="hello_world"*	11

Table 1
DEFINITION OF THE SIZE OF A BUFFER

In the following sections, we present some translation rules to derive a B model from a C program.

5.3 Translation of buffers

To detect buffer overflows, we have to generate information about the maximum size of each declared buffer and also the amount of the data that it contains at any time. This information is stored in two integer B variables *buffer_max* and *buffer_used* defined as follows:

Constants	*buffer_max*
Properties	*buffer_max=size_buff*
Variables	*buffer_used*
Invariant	*buffer_used* $\in$ NAT $\wedge$ *buffer_used* $\leq$ *buffer_max*

where $size\_buff$ is the size of the declared buffer that is defined according to a set of rules described in [16]. Hereafter, Table 1 gives some examples on how $size\_buff$ is obtained.

Variable $buffer\_used$, initialized to 0, is updated each time a new variable is assigned to the buffer. In [16], we have defined a set of rules that generate a B substitution for each C instruction that modifies the value of the buffer. Table 2 gives some examples.

Substitution (**CHOICE** $buffer\_used := 1$ **OR** $buffer\_used := buffer\_max$ **END**) means that we have to deal with limit cases, that is, the buffer may contain data

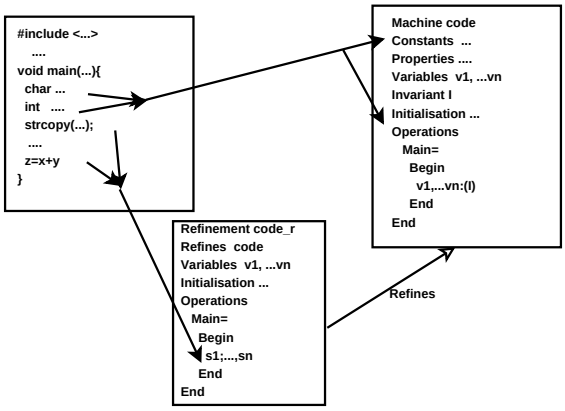


Fig. 2
ARCHITECTURE OF THE B SPECIFICATION

Buffer assignment	**B substitution on** $buffer\_used$
char buff[N]="*hello_world*"	$buffer\_used := 11;$
$gets(buff)$	**CHOICE** $buffer\_used := 1$ **OR** $buffer\_used := buffer\_max$ **END**
$fscanf(file\_name,"\%s",\&buff)$	
$scanf("\%s",\&buff)$	
$fgets(buff, size, fp)$	$buff\_used := size$
$read(fd, buff, size)$	
$strcpy(dest\_buff, source\_buff)$	$dest\_buff\_used := source\_buff\_used$
$strncpy(dest\_buff, src\_buff, size)$	$dest\_buff\_used := size$

Table 2
B TRANSLATION OF C INSTRUCTIONS ON BUFFERS

of minimum or maximum size, this size may be especially equal $buffer\_max$ to simulate a potential attacker. To detect a potential vulnerability at each point of the program, we have to add the following B substitution after each assignment on variable $buffer\_used$

ASSERT ($buffer\_used \leq buffer\_max$) **THEN**
$skip$
END

Type declaration	x_{min}	x_{max}
signed short int x	-32767	32767
unsigned short int x	0	65535
Signed int x	-MAXINT	MAXINT
Unsigned int x	0	2*MAXINT+1

Table 3
MINIMUM AND MAXIMUM VALUES OF SOME C ARITHMETIC TYPES

5.4 Translation of arithmetic data

To detect arithmetic vulnerabilities, we have to verify whether the value assigned to a given variable goes beyond the scope of its type. To do this, we have to memorize the minimum x_{max} and maximum x_{min} values that each variable x can represent. Table 3 gives some examples on how these variables are defined according to the different types of the C language:
For each declared variable x, we generate the following invariant to state that its value must belong to the range of its type: ($x \in x_{min} \,..\, x_{max}$). To detect vulnerabilities, we have also to translate each C instruction that modifies variable x. To get a value from a user, function $scanf()$ is the most common one. Since we have to consider the worst case, each C instruction $scanf("format", \&x)$ is translated into the following B substitutions: **CHOICE** $x := x_{min}$ **OR** $x := x_{max}$ **END**. Other arithmetic operations, like additions, subtractions, multiplications and assignments are straightforwardly translated into B since they are all supported by similar operators with equivalent semantics. As for buffers, after each assignment of variable x, we have to add the following B substitution in order to detect any vulnerability: **ASSERT** ($x \in x_{min} \,..\, x_{max}$) **THEN** $skip$ **END**.

5.5 Translation of C control structures into B

So far, we have seen how to translate declarations (buffer and arithmetic data) and basic C instructions into B. However, a C code may include control structures like conditional (**IF**) and sequential structures. The C control structures we consider in this paper are described by the following BNF grammar:

Instruction ::=
$/* \; assignment \; */$
  | $varID = expression$
$/* \; Sequencing \; */$
  | $Instruction_1 ; Instruction_2$
$/* \; choice \; */$
  | **if** (*expression*) { $Instruction_1$ } **else** { $Instruction_2$ }

Since the B method supports all the above C control structures with the same semantics, the choice, assignment and sequencing instructions are mapped straightforwardly into the **IF**, assignment and sequencing substitutions respectively of the B language.

5.6 Illustrative example

To illustrate our approach, let us apply the different translation rules we have defined on the following C program:

```
#include <stdio.h>
void main(){
    FILE *fp;
    char tmp[40], buff[20];
    fp = fopen("example.txt", "r");
    fgets(tmp,30,fp); strcpy(buff,tmp);
    puts(buff); short int x,y;
    scanf("%h",&x); y=x+2; y=x;}
```

which is translated into:

MACHINE $Example$
CONSTANTS $tmp_{max}, buff_{max}$
PROPERTIES $tmp_{max} = 40 \wedge buff_{max} = 20$
VARIABLES tmp_{used}, $buff_{used}$, x, y
INVARIANT
$tmp_{used} \in NAT \wedge tmp_{used} \leq tmp_{max} \wedge$
$buff_{used} \in NAT \wedge buff_{used} \leq buf_{max} \wedge$
$x \in 0..32767 \wedge y \in 0..32767$
INITIALISATION tmp_{used}, $buff_{used}$, x, $y := 0, 0, 0, 0$
OPERATIONS
Main = **BEGIN**
$tmp_{used}, buff_{used}, x, y :$
$(tmp_{used} \in NAT \wedge tmp_{used} \leq tmp_{max} \wedge$
$buff_{used} \in NAT \wedge buff_{used} \leq buf_{max} \wedge$
$x \in 0..32767 \wedge y \in 0..32767)$
END
END

and

REFINEMENT $Example\_Ref$
REFINES $Example$
VARIABLES tmp_{used}, $buff_{used}$, x, y
INITIALISATION tmp_{used}, $buff_{used}$, x, y:=$0, 0, 0, 0$
OPERATIONS
Main = **BEGIN**
/*translation of fgets(tmp,30,fp)*/
$tmp_{used} := 30;$
ASSERT $(tmp_{used} \leq tmp_{max})$ **THEN** $skip$ **END**;
/*translation of strcpy(buff,tmp)*/
$buff_{used} := tmp_{used};$
ASSERT $(buff_{used} \leq buff_{max})$ **THEN** $skip$ **END**;
/*translation of scanf("%h",&x)*/
CHOICE $x := 0$ **OR** $x := 32767$ **END**;
$y := x + 2;$
ASSERT $(y \in 0..32767)$ **THEN** $skip$ **END**;
$y := x;$
ASSERT $(y \in 0..32767)$ **THEN** $skip$ **END**
END
END

6. Vulnerability detection

In this section, we discuss how it is possible to detect vulnerabilities thanks to refinement proofs but also using the ProB tool. To demonstrate the absence of vulnerabilities, we have to discharge all the specification proofs of the abstract component but also those of the refinement component:

1. *Specification proofs*: these proofs aim at demonstrating that operation $Main$, defined in the first abstract level, is correct. We have to prove that this operation re-establishes the invariant: $Inv \Rightarrow [V_1, \ldots V_n : (Inv)]Inv$ where V_i and Inv denote respectively the variables and the invariant of the abstract component. This proof is obvious and automatically discharged.

2. *Refinement proofs*: these proofs aim at demonstrating that the refinement of operation $Main$ does not contradict or violate its abstraction. In other words, we have to prove that the refined operation also re-establishes the invariant: $Inv \Rightarrow [S]Inv$ where S denotes the B translation of the instructions defined in the C program. When establishing these proofs, two cases are possible:

- the proof is discharged (automatically or interactively): this case means that the program is correct and does not contain any vulnerability.

- the prover of AtelierB fails to discharge the proof: this case means that the prover fails to prove a given goal (Predicate) G. Since the complexity of these proofs is very low, such a failure denotes an invariant violation in general, that is, the occurrence of a vulnerability in the corresponding C program. To be sure that this potential vulnerability is a real one, we use the model checking functionality of the ProB tool as a plugins in order to check that the invariant is really violated by finding, for instance, a state that does not satisfy predicate G or contradicts the global invariant.

Let us illustrate this approach on the previous example. For the abstract component $Example$, the POG of AtelierB generates 3 proof obligations that are all automatically discharged as expected. However for the refinement

component $Example\_Ref$, the POG also produces six proof obligations but it succeeds to perform only five of them. The remaining proof obligation is related to invariant $(buff\_used \leq buff\_max)$: $(30 \leq buff\_max)$ which is obviously false. Let us remark that the prover did not point out that invariant $(yy \in 0..32767)$ is also violated. The reason is that the semantics of the **ASSERT** substitution considers its predicate as true to continuous the remaining substitutions. As predicate $(30 \leq buff\_max)$ is not fulfilled, any invariant becomes true ($false \Rightarrow Inv$ is always true). This feature is very interesting in our case since it permits an incremental vulnerability detection. Now, let us replace substitution $(tmp_{used} := 30)$ by $(tmp_{used} := 10)$. The POG generates the following unproved proof related to invariant $(yy \in 0..32767)$: $(xx + 2 \in 0..32767)$. To be sure that this formula is not valid. We use PROB model checker in order to find a state that satisfies its complement $(xx + 2 \notin 0..32767)$. Figure 3 depicts the answer of PROB model checker for the query of finding a state that satisfies the last predicate. Since, a state that violates our goal $(xx+2 \in 0..32767)$ is found, we can conclude that the instruction $(y := x + 2)$ is vulnerable.

7. Related work

To deal with vulnerabilities, several techniques have been developed. We present hereafter those related to buffers and arithmetic data:

- *Buffer overflows vulnerabilities*: the techniques to detect buffer overflows can fall into two categories: *static* and *dynamic*. According to the dynamic technique, the buffer overflow is avoided by restricting the access to the memory of an application. This method primarily relies on a safe code being preloaded before an application is executed. This preloaded component can either provide safer versions of the standard unsafe functions, or it can ensure that the return addresses of the functions are not overwritten. *StackGuard* [5] is an example of the tools following such a method. It places a canary word next to the return address whenever a function is called. If the canary word has been altered when the function returns, we are sure some attempt has been made on the overflow buffers. In that case, it responds by emitting an alert message and halting the program. This technique is also used by *StackShield*[1], *SFI* [20], *libsafe*[2] and *GCC*. Another approach developed by Arash Baratloo et. al. [2] consists in replacing unsafe library functions with safe implementations. However, run-time solutions always incur some performance penalty(StackGuard reports performance overhead up to 40% [7]). The other problem with run-time solutions is that while they may be able to detect or prevent a buffer overflow attack, they effectively turn it into a denial-of-service attack.

[1]http://www.angelfire.com/sk/stackshield

[2]http://www.research.avayalabs.com/project/libsafe

Static techniques permit to overcome this problem by detecting likely vulnerabilities before deployment. RATS [9] and ITS4 [19] are examples of tools based on the static analysis technique. They use lexical analysis and compare the identified lexemes with a "suspects" database to identify vulnerabilities in C source files. The main drawback of such tools is that they may generate false warnings and sometimes miss real problems. Another tool, CodeAuditor[21], uses Model checking to find the vulnerabilities. However as we know, if the code is very large, we can not finish the verification in an acceptable period of time. If we reduce the code source size, we may miss some vulnerabilities also.

- *Integer vulnerabilities*: one approach to fix integer vulnerabilities is to raise a compile-time warning for each potential vulnerability and let the programmer fix them. However, the number of actual vulnerabilities is an order of magnitude less than the number of warnings. Another approach consists in translating the C code into a type-safe code, e.g., *Cyclone* [10] or *CCured* [14]. However, this option may not be practical in many settings, such as for performance-critical applications or when the user is not familiar with the type-safe code. *PICK* [3] is a tool that uses sub-typing rules to detect unsafe integer operations and inserts the necessary dynamic checks to prevent exploits. But it only deals with the casting vulnerabilities. *RICH* [4] uses a similar approach to *PICK* and also suffers of denial of service attacks.

In this paper, we define a formal approach to detect buffer and arithmetic vulnerabilities using the B method. Our approach consists of a static analysis of the code in order to extract the sufficient and the necessary information about the different elements that may produce vulnerabilities. This information is then modeled in B in order to prove the presence/absence of vulnerabilities. Based on a static technique and a formal method, our approach has the advantage of avoiding denial of service attacks and false alarms at the same time.

8. Conclusion and future work

Software security has always been a concern in the software industry. The C language is one of the most used program language in the world, also its vulnerabilities are well-known. In this paper, we have defined a formal method based on the B method in order to detect vulnerabilities with the ultimate goal of correcting them. Basically, our proposal consists in defining variables to store the size of buffers, and also to store the values each integer variable can represent. These different informations together with the C program are then translated into a B specification on which we perform proof and model checking activities to detect the presence of vulnerabilities. We have applied this approach on several applications (larger and more complex than the example given in this paper) that gave very promising results.

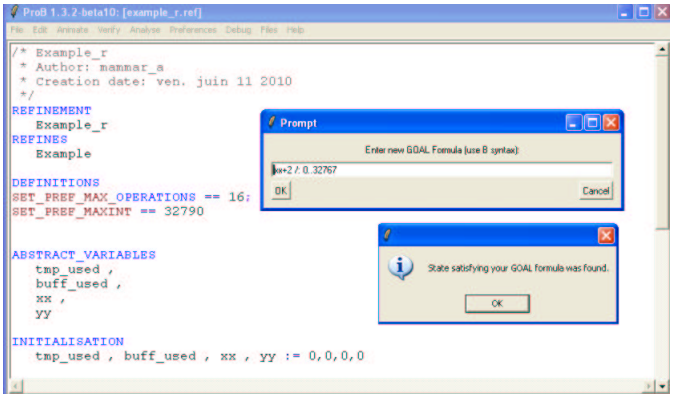


Fig. 3
DETECTION OF VULNERABILITIES USING PROB MODEL CHECKER

Compared to other approaches, our approach can detect buffer overflow and Integer vulnerabilities at the same time without generating false alarms. In addition, our proposal can be applied on any other programming language, like JAVA for instance, by defining new translation rules to deal with its vulnerable functions. Finally to make our approach workable, we have developed a tool that permits us to point out the vulnerable statements in a C program. This tool is implemented in JAVA and described elsewhere in [16]. Currently, we are working on extending the proposed approach to support loops and recursive functions. Future work include also the development of a process that would permit to correct automatically the C code by exploring the B proof and model-checking result activities. Basically, we have to define the condition to add within the $Main$ operation in order to establish the invariant and correct the corresponding C program. To do that, we can reuse our previous work presented in [13] that defines a formal process to calculate the weakest precondition of a B operation to re-establish an invariant.

References

[1] ABRIAL J. R.: The B-Book: Assigning Programs to Meanings, Cambridge University Press, (1996).

[2] Baratloo, A., Singh, N., Tsai, T.: Transparent Run-Time Defense Against Stack-Smashing Attacks Protecting Systems from Stack Smashing Attacks with StackGuard, the 9^{th} USENIX Security Symposium, (2000).

[3] Brumley, D., Song, D., Slember, J.: Towards Automatically Eliminating Integer-Based Vulnerabilities, Technical Report, School of Computer Science, Carnegie Mellon University, (2006).

[4] Brumley, D., Xiaodong, D., Song and Tzi-cker Chiueh and Rob Johnson and Huijia Lin RICH: Automatically Protecting Against Integer-Based Vulnerabilities, the 14^{th} Annual Network & Distributed System Security Symposium(NDSS), (2007).

[5] Calton, C.C., Pu C., Maier D., Hinton H., Walpole, J., Bakke, P., Beattie S., Grier, A., Wagle, P., Zhang, Q.: StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks, the 7^{th} USENIX Security Symposium, (1998).

[6] CERT Coordination Center. CERT/CC statistics http://www.cert.org/stats/historical.html, (2007).

[7] Cowan, C., Beattie, S., Day, R.F., Pu, C., Wagle, P., Walthinsen, E.: Protecting Systems from Stack Smashing Attacks with StackGuard, the 7^{th} USENIX Security Symposium, (1998).

[8] Cowan, C., Barringer, M., Beattie, S., Kroah-Hartman, G.: FormatGuard : Automatic Protection From printf Format String Vulnerabilities, the 10^{th} USENIX Security Symposium, (2001).

[9] Fortify Software Inc Secure software solutions, rats, the rough auditing tool for security, http://www.securesw.com/rats, (2001).

[10] Jim, T., Morrisett, J.G., Grossman, D., Hicks, M.W. Cheney, J., Wang, Y.: Cyclone: A Safe Dialect of C, the 9^{the} USENIX Annual Technical Conference, (2002).

[11] Jimenez, W., Mammar, A., Cavalli, A.R.: Software Vulnerabilities, Prevention and Detection Methods: A Review, the 1^{first} International Workshop on Security in Model Driven Architecture(SEC-MDA), (2009).

[12] Kuang, C., Miao, Q., Chen, H.: Analysis of Software Vulnerability, the 5^{th} WSEAS International Conference on Information Security and Privacy(ISP), (2006).

[13] X, Y.: A Systematic Approach to Generate B Preconditions: Application to the Database Domain, Software and System Modeling 8(3), (2009).

[14] Necula, G., McPeak, S., Weimer, W.: CCured: Type-Safe Retrofitting of Legacy Code, the 29^{th} Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), (2002).

[15] NIST. The Economic Impacts of Inadequate Infrastructure for Software Testing. National Institute of Standards and Technology-Final Report, (2002).

[16] X, Y.: Detecting C Program Vulnerabilities Master Thesis, Telecom SudParis, (2009). Available at http://phoenix.inria.fr/index.php/members/36-pengfei-liu.

[18] Shankar, U., Talwar, K., Foster, J.S., Wagner, D.: Detecting Format String Vulnerabilities with Type Qualifiers, the 10^{th} conference on USENIX Security Symposium(SSYM), (2001).

[19] Viega, J., Bloch, J. T., Kohno, Y., McGraw, G.: ITS4: A Static Vulnerability Scanner for C and C++ Code, the 16^{th} IEEE Annual Conference Computer Security Applications(ACSAC), (2000).

[20] Wahbe, R., Lucco, S., Anderson, T.E., Graham, S.L.: Efficient Software-Based Fault Isolation, the 14^{th} ACM Symposium on Operating Systems Principles(SOSP), (1993).

[21] Wang, L., Zhang, Q., Zhao, P.: Automated Detection of Code Vulnerabilities Based on Program Analysis and Model Checking, the 8^{th} IEEE International Working Conference on Source Code Analysis and Manipulation(SCAM), (2008).

[22] ProB.: http://users.ecs.soton.ac.uk/mal/systems/prob.html.

The Edge-Pushing LR(k) Algorithm

X. Chen[1], **D. Pager**[1]
[1] Department of Information and Computer Science, University of Hawaii at Manoa, Honolulu, HI, USA

Abstract - *The original LR(k) parser generation algorithm of Knuth in 1965 is very expensive in time and space. Different approaches have been proposed to achieve LR(k) with better practical performance. This work designed a new LR(k) algorithm called the edge-pushing algorithm, which is based on recursively applying the lane-tracing process. The algorithm has been implemented into the parser generator Hyacc. Here we first present the background and related work on LR(k) parser generation, next we introduce the edge-pushing algorithm's design and implementation, its LR(k) parse engine and corresponding storage parsing table. Relevant issues are discussed. Finally we give some applicable LR(k) grammar examples*.

Keywords: LR(k), Parser Generation, Edge-Pushing, Lane-Tracing, Algorithm

1 Introduction

The canonical LR(k) algorithm proposed by Knuth in 1965 [1] is a very powerful parser generation algorithm for context-free languages. However it is too expensive in time and space costs to be practical. Subsequent research on reduced-space LR(1) algorithms, which can reduce the state space and thus improve the performance of canonical LR(1) parser generation, were made by researchers such as Pager [2][3][4] and Spector [5][6]. LR(k) parser generation is in general much more expensive and complicated than LR(1) parser generation. Although widely studied on the theoretical side, very little practical work has been done due to the performance problem. It would be of value to study practical LR(k) parser generation algorithms. Here we show the design and implementation of a LR(k) algorithm called the edge-pushing algorithm, which is based on the lane-tracing LR(1) algorithm. The edge-pushing algorithm has been implemented into the parser generator Hyacc [7][8].

In this discussion, Greek letters such as α, β, γ, ψ, φ, ω ... represent a string of symbols. ε represents the empty string. In the context of specifying a grammar, Roman letters such as A, B, C, ..., a, b, c, ... represent a single symbol; of these upper case letters represent non-terminal symbols, and lower case letters represent terminal symbols. In the context of specifying algorithms or mathematical formula, Roman letters may have other meanings such as a string, a number, a state or a set, and are not limited to terminal or non-terminal symbols. The symbol ⊣ stands for the end of an input stream, and Ø stands for the empty set. The concepts of *state*, *configuration* and *theads*(*α*, *k*) used here are equivalent to "item set", "item" and $FIRST_k(\alpha)$ correspondingly in some other literature.

2 Background And Related Work On LR(k) Parser Generation

The 1965 paper of Knuth [1] introduced LR(k) parser generation. After that, a lot of works were done to reduce performance cost.

The work of Pager in the 1970s [2][3][4] were about reduced-space LR(k) parser generation. Pager reported LR(k) analysis on the grammars of real languages such as ALGOL for LR(2) and LR(3) [3]. M. Ancona et. al. published their work on LR(k) parser generation from 1980s to 1990s [9][10][11][12][13]. They proposed a method [13] in which non-terminals are not expanded to terminals in contexts, and expansion is not done until absolutely needed to resolve inadequacy. This defers the calculation of $FIRST_k(\alpha)$ until absolutely necessary. They claim savings in both time and storage space by deploying this method when trying on several programming language grammars. They have worked on a LR(k) parser generator for their research, but no publicly available product was reported. In 1993, Terence Parr's PhD thesis "Obtaining practical variants of LL(k) and LR(k) for k > 1 by splitting the atomic k-tuple" [14] provided important theoretical implications for working on multiple lookaheads and claimed close-to-linear approximation to the exponential problem. The idea is to break the context k-tuples, which can be applied to both LL(k) and LR(k). Parr's ANTLR LL(k) parser generator was a big success. LL(k) parser generation is considered easier to work with. Theoretically it is also less powerful than LR(k) in recognition power. Parr's PhD thesis proposed that adding semantic actions to a LR(k) grammar degrades its recognition power to that of a LL(k) grammar. Based on this proposition he worked on LL(k) parser generation only. Josef Grosch worked on a LR(1)/LR(k) parser generator in 1995 [15]. In case of LR(1), it was practical only for small to medium size grammars, and LR(k) is certainly more expensive. Bob Buckley worked on a LR(1) parser generator called Gofer in 1995 [16]. He said it was a long way to go from being a production software. More recently in 2005, Karsten Nyblad claimed to have a plan for a LR(k) implementation [17]. There was no more news so far. Chris Clark worked on a LALR(k)/LR(1)/LR(k) parser generator

Yacc++ [18][19]. Its LR(1)/LR(k) implementation was loosely based on Spector's paper [5][6]. But there was an infinite loop problem on LR(k). Thus the LR(k) feature of Yacc++ was only used internally and did not go public. Ralph Boland once worked on LR(k) [20], but report on his results was not found. Paul Mann mentioned that Ron Newman's Dr. Parse [21] works on LR(k) for k = 2 or maybe 3. The only claimed successful efficient LR(k) parser generator is the MSTA parser generator in the COCOM tool set [22]. The author Vladimir Makarov said it generates fast LALR(k) and LR(k) grammar parsers with "acceptable space requirements".

To conclude, practical result on LR(k) parser generation is scarce.

3 LR(k) Parser Generation Based On Edge-Pushing

This section discusses practical LR(k) parser generation based on the edge-pushing algorithm as implemented in Hyacc.

The exponential behavior of LR(k) parser generation comes from two sources: 1) the number of states in the parsing machine, and 2) the number of context tuples of the configurations. The reduced-space LR(1) algorithms such as those by Pager can solve the first problem. The second problem can be solved following the way of Terence Parr [14], or as in the edge-pushing algorithm discussed here by only working on those configurations that actually lead to reduce/reduce conflicts and ignore the rest. The edge-pushing algorithm does this by recursively applying the lane-tracing procedure.

Three problems need to be solved when extending lane-tracing to LR(k) in a practical way: 1) LR(k) algorithm. This part should extend the lane-tracing LR(1) parser generation algorithm, such that it can be recursively applied to states where reduce/reduce conflicts cannot be resolved by available lookaheads. 2) Storage of LR(k) parsing table. This part should efficiently represent the LR(k) lookaheads in LR(k) parsing table and work together with the LR(k) algorithm in 1). 3) LR(k) parse engine. After the LR(k) parsing table is generated, the Hyacc LR(1) parse engine should be extended so it can use the LR(k) parsing table for parsing LR(k) grammars.

3.1 The Edge-Pushing LR(k) algorithm

For the ease of discussion, we define the terms and functions below. A *final configuration* is one where the marker (the dot) is at the right most position on the right hand side, e.g., E → E + T ▪. A *head configuration* is a configuration at the start of a lane where we stop in lane-tracing. On the contrary, by *tail configuration* we mean those configurations at the end of a lane from which we start the lane-tracing. Define the function *theads(α, k)* to return a set of terminal strings obtained from α. The length of these strings is k, and this function is potentially exponential on k. *theads(α, k)* is the same as $FIRST_k(\alpha)$ in many other literature.

3.1.1 LR(k) parser generation based on recursive lane-tracing

Each time after LR(k) lane-tracing for a certain k, we need to check if conflicts are resolved, and further trace LR(k+1) only for states with unresolved reduce/reduce conflicts. In order to resolve conflicts, we may need to go back multiple levels. The purpose here is to trace back all the way until we find relevant contexts of length k that solve the reduce/reduce conflicts of inadequate states. The word "relevant" here means we only get contexts that are useful for resolving conflicts: only for those contexts that cause conflicts, we trace further. This is needed because the number of LR(k) contexts can increase exponentially with k. We also should better cache the computation of LR(k) theads for more efficient computation of LR(k+1) theads.

It is possible to do LR(k) lane-tracing on a specific k for each inadequate state, then increase k and do this again on all unresolved states. It is also possible to do LR(k) lane-tracing recursively on one inadequate state, increase k on this state only until its inadequacy is resolved or found not resolvable, then start on another inadequate state. For these two methods, the second may be easier from a practical point of view, since if we want to take advantage of the result of LR(k) for LR(k+1), the second method allows us to remember less intermediate information.

Below is an intuitive and straightforward solution for LR(k) lane-tracing. Algorithm 1 *Conflicts_Resolved(S, k)* checks if the reduce/reduce conflicts on a state S can be solved by LR(k) lane-tracing. Algorithm 2 *Intuitive_Lane_Tra-cing_LRk(S)* achieves LR(k) by calling Algorithm 1 and increasing the parameter k from 2 to 3, 4, … until *Conflicts_Resolved(S, k)* returns true. The problem of Algorithm 2 is that it always repeats the computation for each LR(k) context, even for those configurations that do not have associated conflict. E.g., suppose two reduction configurations r1 and r2 both have conflicted LR(1) context {'a'}. Then for the LR(2) context, r1 has context set {'ab', 'ac'}, r2 has context set {'ab', 'ad'}. Then we only need to continue with LR(3) on the configurations that generate conflict 'ab'. But Algorithm 2 also computes the configurations that generate 'ac' and 'ad'. To avoid this problem and achieve more delicate control, we can improve by only computing the configurations that generate 'ab'. We call such a method "edge-pushing" and show it below in a conceptual example.

Algorithm 1: Conflicts_Resolved(S, k)

```
1 foreach final configuration Cf of state S do
2     Cs ← head configuration of Cf;
3     get LR(k) context of Cs;
4 if all reduce/reduce conflicts are resolved then
5     add context obtained to parsing table;
6     return true;
7 else
8     return false;
```

Algorithm 2: Intuitive_Lane_Tracing_LRk(S)

```
1 k ← 2;
2 repeat
3       resolved ← Conflicts_Resolved(S,
k);
4     k ← k + 1;
5 until resolved == true
```

3.1.2 Edge-pushing – a conceptual example

Below we will show a conceptual description of LR(k) lane-tracing with edge-pushing.

Here the starting state is state 10 with a LR(1) reduce/reduce conflict. Four steps are carried out from LR(2) to LR(5) to resolve the conflict eventually. In each graph, the circles with bold edge are at the cutting edge of lane-tracing. Circle 10 stands for state 10. Circles 3 to 9 actually should mean a head configuration in states 3 to 9, and here by saying state 3 we actually mean a head configuration in state 3. z is a local variable associated with a head configuration, whose value is obtained by adding together z and k' of the tail configuration. k' is the value used in the calculation of *theads(β, k')* for a configuration A → α • B β. $k' = k - z$, where k is the k in LR(k), and z is the local z. The figures below show a conceptual example of the calculation of these values, and how LR(k) lane-tracing is pushed at the cutting edge.

Fig. 1 shows the initial conflict state obtained by LR(1) lane-tracing:

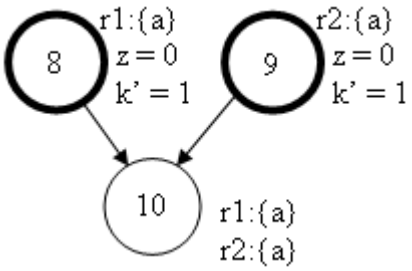


Fig. 1. LR(1) state with reduce/reduce conflict.

In Fig. 2, states 8 and 9 are at the cutting edge of lane-tracing. For state 8, on its right side "r1: {ab, ac}" means that 'b' and 'c' are the context symbols generated by a configuration in state 8 for reduction r1. Local $z = 0$. It is always 0 for *head configuration*s in LR(1) and LR(2) lane-tracing. The value of local k' is obtained by subtracting the local z from the k in LR(k): $k'_8 = k - z_8 = 2 - 0 = 2$. The meanings of notations are similar for state 9.

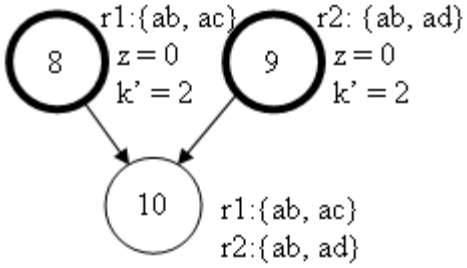


Fig. 2. LR(2) states.

In Fig. 3, states 5, 6 and 7 are at the cutting edge of LR(3) lane-tracing. The only thing that needs explanation is the value of z. For state 5, z is obtained by the sum of k' and z in its tail configuration in state 8: $z_5 = k'_8 + z_8 = 2 + 0 = 2$. Similarly z is obtained this way in states 6 and 7.

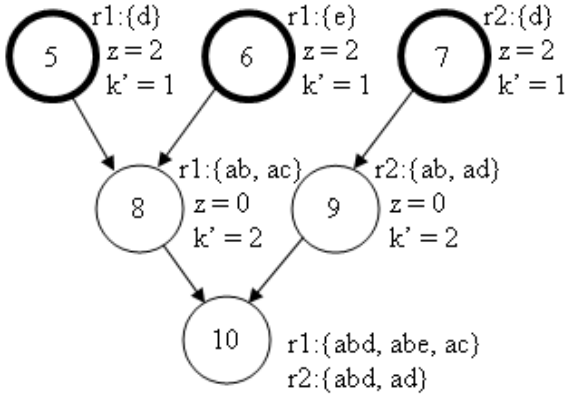


Fig. 3. LR(3) states.

See Fig. 4, in state 5 we get two consecutive context symbols "dd" by doing *theads(β, k'5)* calculation where $k'_5 = k - z_5 = 4 - 2 = 2$. In state 7, $k'_7 = k - z_7 = 4 - 2 = 2$, *theads*($\beta$, k'_7) returns 'd' whose length is less than 2, so we have to do lane-tracing here to obtain state 4 as shown. In the head configuration in state 4 $z_4 = k'_7 + z_7 = 2 + 1 = 3$, $k'_4 = k - z_4 = 4 - 3 = 1$, and we do *theads*($\beta$, k'_4) to obtain context 'd'.

In Fig. 5, states 3 and 4 are at the cutting edge of lane-tracing, and we obtain the values of z and k' for them in the same way as before: $z_3 = z_5 + k'_5 = 2 + 2 = 4$, $k'_3 = k - z_3 = 5 - 4 = 1$. $z_4 = 3$ was obtained in the last step, $k'_4 = k - z_4 = 5 - 3 = 2$.

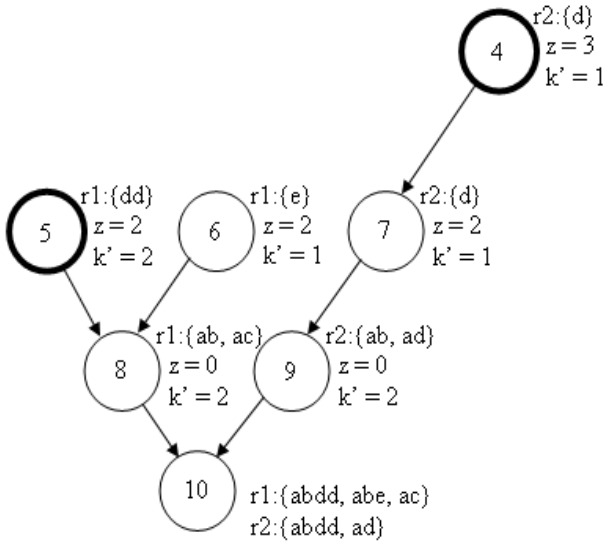


Fig. 4. LR(4) states.

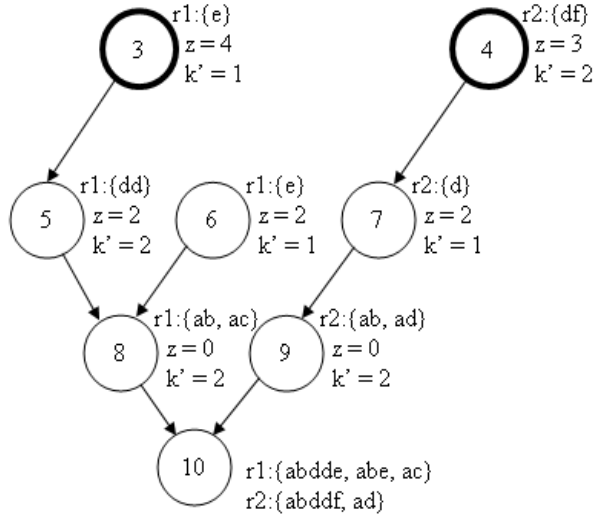


Fig. 5. LR(5) states.

Now, we don't need to add any context to state 10's final configurations, because the LR(k) parsing tables (LR(1) parsing table, LR(2) parsing table, ..., LR(5) parsing table) suffice for both storage of contexts as well as conflict detection.

LR(k) parsing tables 1 to 5 are the corresponding LR(1) to LR(5) parsing tables. Symbol Θ means a reduce/reduce conflict. See section 3.3 for the exact notations on storage of LR(k) parsing tables. These tables are created when we do the LR(k) lane-tracing. Whenever a conflict is found: e.g., when inserting action r2 to a field we find an r1 action already exists in the same cell, then we know a conflict occurs, and then we pass the two relevant configurations to the next round of lane-tracing.

Table 1. LR(1) parsing table.

state / token	…	a	…
…			
10		Θ	
…			

Table 2. LR(2) parsing table.

(state, LR(1) lookahead) / token	b	c	d
(10, a)	Θ	r1	r2

Table 3. LR(3) parsing table.

(state, LR(2) lookahead) / token	d	e
(10, b)	Θ	r1

Table 4. LR(4) parsing table.

(state, LR(3) lookahead) / token	d
(10, d)	Θ

Table 5. LR(5) parsing table.

(state, LR(4) lookahead) / token	e	f
(10, d)	r1	r2

When parsing an input string, we follow the LR(1), ... LR(k) parsing tables to find a match. Suppose during a parse we are in state 10, and the next few lookaheads of the input string are "abddf". The first lookahead symbol 'a' gets us a Θ action which denotes a reduce/reduce conflict, so we take the second symbol 'b' and go to the LR(2) parsing table. There we find another Θ action so need to take the third symbol 'd' and go to the LR(3) parsing table. This chain of actions stops at the LR(5) parsing table, where the 5th lookahead 'f' denotes an action 'r2', which means to reduce by rule 2.

3.1.3 The Edge-pushing algorithm

We summarize the skeleton of the edge-pushing algorithm below as Algorithm 3.

There are lots of details involved, but two major components are lane-tracing and calculation of *theads(α, k)*. The edge-pushing algorithm uses iteration and avoids recursion. Two configuration sets are used between iterations, such that during a round of iteration, we draw an element from the working set (Set_C), process it and add new derived configurations to the derived set (Set_C2); then at the end of the iteration, pass the elements of the derived set to the working set and start the next iteration. The edge-pushing algorithm stops when lane-tracing is back to state 0, line 16 "Σ ← lane_tracing(C)" will return an empty set. So eventually Set_C2, and thus Set_C, becomes an empty set.

Algorithm 3: Edge_Pushing(S)

```
Input: Inadequate state S

1  Set_C ← Ø; Set_C2 ← Ø
2  k ← 1;
3  foreach final configuration T of S do
4     T.z ← 0;
5     Let C be the head configuration of T,
        and X be the context generated by C;
6     Add triplet (C, X, T) to set Set_C;
7  while Set_C ≠ Ø do
8     k ← k + 1;
9     foreach (C:A → α • B β, X, T) in Set_C do
10      k' ← k - C.z;
11      calculate ψ ← theads(β, k');
12      foreach context string x in ψ do
13        if x.length == k' then
14          Insert (S, X, last symbol of
              string x, C, T) to Set_C2
              and add to LR(k) parsing table;
15        else if x.length == k' - 1 then
16          Σ ← lane_tracing(C);
            //Σ: set of head configurations
17          foreach configuration σ in Σ do
18            σ.z ← C.z + k';
19            Let m be the generated context
                symbol in σ;
20            Insert(S, X, m, σ, T) to Set_C2
              and add to LR(k) parsing table;
21    Set_C ← Set_C2;
22    Set_C2 ← Ø;
```

At the beginning we initialize the variable *z* of relevant final configurations to 0, and then obtain the *z* values for derived configurations recursively. Each time lane-tracing is done, we only use the LR(1) theads of the new head configurations. This is easier. In comparison, in Algorithm 1 when lane-tracing is involved, we may need to go several rounds of lane-tracing recursively to get all the LR(k) theads, which is much harder. We do not need to attach any context to the initial inadequate states' final configurations, because here the LR(k) parsing tables can store the LR(k) context symbols as well as detect possible conflicts. Here since we only push the cutting edge of lane-tracing, we avoid recalculating those edges that do not cause conflict.

Due to the exponential nature of *theads(α, k)*, we potentially still may have an exponential problem. However, for the entire parsing machine, the number of inadequate states is small. Further, those configurations that cause conflicts for increasing k may be just a portion of all such initial configurations, i.e., configurations that we should trace further for LR(k+1) usually are just a small portion of the configurations that we trace for LR(k). Thus we should expect a below exponential increase in most cases.

3.1.4 Edge-pushing algorithm on cycle condition

We have described the edge-pushing algorithm on one state. Complication is involved when lane-tracing of different inadequate states come into the same configurations (e.g. Fig. 6 (a)), or when cycles are involved in the tracing (e.g. Fig. 6 (b)).

Using Figure 6 (b) as an example, if LR(k) tracing ends at state B and cannot resolve the reduce/reduce conflict, LR(k+1) tracing ends at state C and cannot resolve the conflict, LR(k+2) tracing ends at state D and still cannot resolve the conflict, then LR(k+3) tracing will come back to state B. This forms an infinite cycle: B → C → D → B …

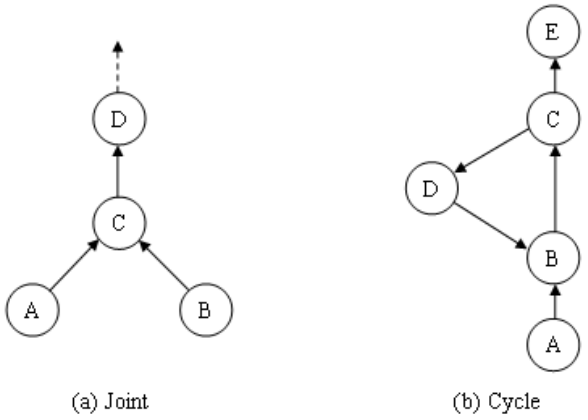


Fig. 6. LR(k) lane-tracing: joint and cycle conditions.

Then we need to answer two questions: 1) Do we need to cache a previous computation? 2) What to do with the parameters *k'* and *z*? For 1), it is obvious that caching a previous computation has advantages. Once we do cache, then we also don't need to worry about 2), since we can refer to the cache for contexts generated. We do not need to care about *k'* and *z*, since these are used only if we do the computation. So every time after lane-tracing and we obtain a set of head configurations, we search in the cache. If it already exists in the cache, then we get the contexts generated from the cache, and also the next round of head configurations from the cache. The problem of cycles can be solved by cache this way. The only left issue is cycle detection: how to avoid tracing down the cycle infinitely. This will be among the future work.

3.2 Computation of theads(α, k)

The *theads*(*α*, *k*), i.e., $FIRST_k(\alpha)$ algorithm used in the edge-pushing algorithm is given by Pager [23]. Compared to the $FIRST_k(\alpha)$ algorithm of Aho and Ullman [24], which uses a bottom-up process, this algorithm takes a top-down approach.

3.3 Storage of LR(k) parsing table

In Hyacc, for an LR(k) grammar, there is an array of parsing tables for each of LR(1), LR(2), ..., LR(k). When resolving conflicts, if the LR(i) parsing table can't do it, we consult the LR(i+1) parsing table. This goes on until the conflict is resolved.

In the LR(k) parsing table ($k \geq 2$), each column represents a lookahead token as in the LR(1) parsing table. Each row represents a (state, token) pair, where the token is a lookahead token that causes reduce/reduce conflict in LR(k-1) parsing table. By doing this we avoid repeating lookaheads for LR(1), LR(2), ... LR(K-1) in the LR(k) parsing table, and can save space.

The Hyacc parser generator uses the hyaccpar file for LR(1) parse engine [7]. For the LR(k) extension, Hyacc uses another parse engine file hyaccpark, which is based on hyaccpar with extension. Table 6 shows the additional variable and arrays used to represent the LR(k) parsing tables besides those for LR(1) in hyaccpark.

In additional, the LR(1) part of the parsing table should have this modification: wherever a reduce/reduce conflict occurs, the previous default reduction number should be replaced by a special value (Θ) labeling the occurrence of a reduce/reduce conflict. In the hyaccpark parse engine's arrays, this would mean the update of corresponding values in arrays yyfs[] and yyptblact[].

Table 6. LR(k) parse engine arrays

Array name	Explanation
yy_lrk_k	The maximum value of k for this LR(k) grammar.
yy_lrk_rows[]	The number of rows in each LR(k) parsing table. For each parsing table, there may be multiple states, and each state may have multiple tokens.
yy_lrk_cols[]	Each row starts with two fields for each (state, token) pair, followed by one entry for each token.
yy_lrk_r[]	The actual values in each LR(k) parsing table. The first two entries are for the (state, token) pair. This is followed by (index, value) pairs, where index is the index of a token in the yyPTC[] array, and value is the action for this token: i) -2 means reduce/reduce conflict, ii) a positive number means no conflict and is the corresponding production's ruleID, iii) -1 labels the end of each row.
yyPTC[]	The values of parsing table column tokens.

3.4 LR(k) parse engine

In Hyacc, the LR(k) parse engine is an extension to the LR(1) parse engine [7]. In the LR(1) parse engine, the action depends on the value at parsing table entry (S, L) in the LR(1) parsing table, where S is a state and L is a lookahead symbol. For LR(k) the only change to the LR(1) parsing table is the addition of the Θ symbol, which leads to the following extension. The LR(k) parse engine extension is shown in Algorithm 4 below. The while loop eventually ends when a reduction is found, either the reduction resolves the reduce/reduce conflict, or the reduction is the end of LR(k) lane-tracing in state 0 and a default reduction is used.

Algorithm 4: LR(k) parse engine extension.

```
Input: Current state S;
       LR(1) lookahead L;
       Action A: action(S, L)

1  if A == Θ then
2      k = 2;
3      while true do
4          L_next ← next lookahead;
5          if L_next == EOF then
6              Report error and exit;
7          else
8              In LR(k) parsing table, find
                 entry A_next ← ((S, L), L_next);
9              if A_next == Θ then
10                 L ← L_next;
11                 k ← k + 1;
12             else
14                 do reduce;
15                 break out of while loop;
```

3.5 Examples

The edge-pushing algorithm has been tried and works on the following grammars. These grammars are often used as examples in LR(k) discussion.

LR(k) grammar G1: S → a A D a | b A D b | a B D b | b B D a, A → a, B → a, D → c^n. Here k depends on the value of n: k = n + 1. c^n is the concatenation of n letters 'c'.

LR(3) grammar G2: S → a A D a | b A D b | a A D b | b C E, A → a, B → a, D → e d, C → B e, E → d a

LR(2) grammar G3: S → a A a | b A b | a B b | b B a, A → C a, B → D a, C → a, D → a

LR(3) grammar G4: S → a A a a | b A a b | a B a b | b B a a, A → C a, B → D a, C → a, D → a

4 Conclusions

We have presented a new LR(k) parser generation algorithm called the edge-pushing algorithm, which is based on the lane-tracing process, recursively traces back on relevant configurations to obtain more contexts to resolve reduce/reduce conflicts. The edge-pushing algorithm, its corresponding LR(k) storage arrays and parse engine have all been designed and implemented into the open source parser generator Hyacc. The edge-pushing algorithm so far works on LR(k) grammars where lane-tracing upon increasing k does not form a cycle.

5 References

[1] Donald E. Knuth. On the translation of languages from left to right. Information and Control, 8(6):607 – 639, 1965.

[2] David Pager. The lane tracing algorithm for constructing LR(k) parsers. In Proceedings of the fifth annual ACM symposium on Theory of computing, pages 172 – 181, Austin, Texas, United States, 1973.

[3] David Pager. The lane-tracing algorithm for constructing LR(k) parsers and ways of enhancing its efficiency. Information Sciences, 12:19 – 42, 1977.

[4] David Pager. A practical general method for constructing LR(k) parsers. Acta Informatica, 7:249 – 268, 1977.

[5] David Spector. Full LR(1) parser generation. ACM SIGPLAN Notices, pages 58 – 66, 1981.

[6] David Spector. Efficient full LR(1) parser generation. ACM SIGPLAN Notices, 23(12):143 – 150, 1988.

[7] Xin Chen. Measuring and Extending LR(1) Parser Generation. PhD thesis, University of Hawaii, August 2009.

[8] Xin Chen. LR(1) Parser Generator Hyacc, January 2008. http://hyacc.sourceforge.net.

[9] Massimo Ancona, Alessandro Paone. Table merging by compatible partitions for LR parsers is NP-complete. Elektronische Informationsverarbeitung und Kybernetik, 30(3):123 – 134, 1994.

[10] Massimo Ancona, Vittoria Gianuzzi. A new method for implementing LR(k) tables. Inf. Process. Lett., 13(4/5):171 – 176, 1981.

[11] Massimo Ancona, Claudia Fassino, Vittoria Gianuzzi. Optimization of LR(k) "Reduced Parsers". Inf. Process. Lett., 41(1):13 – 20, 1992.

[12] Massimo Ancona, Gabriella Dodero, Vittoria Gianuzzi. Building collections of LR(k) items with partial expansion of lookahead strings. SIGPLAN Notices, 17(5):24 – 28, 1982.

[13] Massimo Ancona, Gabriella Dodero, Vittoria Gianuzzi, M. Morgavi. Efficient construction of LR(k) states and tables. ACM Trans. Program. Lang. Syst., 13(1):15 – 178, 1991.

[14] Terence Parr. Obtaining practical variants of LL(k) and LR(k) for k > 1 by splitting the atomic k-tuple. PhD thesis, Purdue University, August 1993.

[15] Josef Grosch, 1995. http://compilers.iecc.com/comparch/article/95-04-179

[16] Bob Buckley, 1995. http://compilers.iecc.com/comparch/article/95-05-087

[17] Karsten Nyblad, 2005. http://compilers.iecc.com/comparch/article/05-03-117

[18] Chris Clark. Yacc++ historical notes, 2005. http://compilers.iecc.com/comparch/article/05-06-124

[19] Chris Clark. More Yacc++ historical notes, 2000. http://compilers.iecc.com/comparch/article/00-02-142

[20] Ralph Boland, 1997. http://compilers.iecc.com/comparch/article/97-11-011

[21] Parser Generator Dr. Parse. http://www.downloadatoz.com/software-development_directory/dr-parse

[22] Vladimir Makarov. Toolset COCOM & scripting language DINO, 2002. http://sourceforge.net/projects/cocom.

[23] David Pager. Evaluating Terminal Heads Of Length K. Technical Report No. ICS2009-06-03, University of Hawaii, Information and Computer Sciences Department, 2008. http://www.ics.hawaii.edu/research/tech-reports/terminals.pdf/view.

[24] Alfred V. Aho, Jeffrey D. Ullman. The Theory of Parsing, Translation and Compiling, Vol. 1, Parsing. Prentice-Hall, Englewood Cliffs, N.J, 1972.

Weighted Method Signatures Fingerprints

S. Leblanc[1] and D. Deugo[2]
[1]School of IT and Engineering, University of Ottawa, Ottawa, Ontario, Canada
[2]The School of Computer Science, Carleton University, Ottawa, Ontario, Canada

Abstract - *A software fingerprint is a set of distinctive characteristics used to identify and compare programs. This paper presents a fingerprint technique based on the characteristics of method signatures. The proposed technique considers uncommon method signatures as being more effective than common ones to compare programs. The implementation of this technique is limited to Java programs, but it could be implemented for other programming languages as well. The experiments carried out to evaluate the technique demonstrated that it was credible, resilient and scalable.*

Keywords: Copyright, Fingerprint, Birthmark, Software theft detection, Java

1 Introduction

Detecting software copyright infringement is challenging because it is difficult to establish if there is a copy relationship between two programs. It follows that the copyright laws are rarely enforced even if the software copyright infringement has a huge economic impacts on the world of software development. This paper presents a technique that can be used to determine if a program is likely to be a copy of another. Consequently, the technique is a useful tool to detect copyright infringement.

1.1 Problem

The root cause of the problem is the impossibility to define what makes two pieces of software equal. Comparing something with a natural and durable identifier is simple. For instance, a bank account has an account number. An account number is a natural and durable identifier of an account because it will not change over time and will always refer to the same account. When two things do not have natural durable identifiers, they are usually compared according to their characteristics. For instance, there are no such things as date IDs (i.e. there are no natural durable identifiers for dates as is the case with bank accounts), but two dates with the same year, month and day are considered to be the same.

Unfortunately, programs do not possess any durable and unique identifiers and the intrinsic nature of all their characteristics is to change over time. Two versions of the same program do not have the same characteristics, but they are still considered to be the same program. On top of that, applying semantics-preserving code transformations, also known as obfuscation, can substitute some characteristics of a program with equivalent ones. Developers who do not want to share their source code often use these transformations to make the source code of their program almost impossible to read. Such transformations are an essential tool to protect the intellectual property of developers. On the other hand, these transformations can also act as a camouflage for developers who infringe copyrights. If a program is difficult to read, then it is also difficult to compare it with other programs. In this context, detecting software copyright infringement is not a simple task.

1.2 Goals

The proposed technique must be able to measure how likely a program is to be a copy of another and must exhibit the five following properties:

Credibility

"Let *p* and *q* be independently written sets of modules which may accomplish the same task. We say *f* is a credible measure if *f(p) != f(q)*" [1].

Resilience to Natural Change

Let *p0* be a set of modules obtained by creating a derivative work from *p*. We say that f is resilient to natural change if *f(p) = f(p0)*.

Resilience to Semantics-Preserving Transformations

"Let *p0* be a set of modules obtained from p by applying semantics-preserving transformations *T*. We say that f is resilient to *T* if *f(p) = f(p0)*" [1].

Running Time Scalability

Let *n* be the size of the set of modules *p*, we say that the running time of *f* is scalable if the average case running time complexity of *f(p)* is at least *O(n)* (linear).

Memory Usage Scalability

Let *n* be the size of the set of modules *p*, we say that the memory usage of *f* is scalable if the average case memory usage complexity of *f(p)* is at least *O(n)* (linear).

1.3 Objectives

Credibility, resilience and scalability cannot be completely reached simultaneously because they are conflicting goals. There are no doubts that considering only completely identical programs as being in a copy relationship would be credible. Such a technique is not likely to produce false positives. On the other hand, this technique would not be resilient. The slightest transformation would prevent this technique from working properly. Also, a simplistic technique that only compares the size of the files of two programs would be able to compare two large programs as effectively as two small ones but would be neither credible nor resilient. The

first objective of the proposed technique is to provide good trade-offs between these conflicting goals.

Its second objective is to be cohesive. Techniques based on a specific measure (as opposed to those that summarize many kinds of measures) are easier to use in conjunction with other techniques. Since it is almost impossible for a technique to meet credibility, resilience and scalability simultaneously, being able to use the proposed technique easily with other existing techniques is definitely an asset.

1.4 Outline

The remainder of this paper is organized as follows. In Section 2, the proposed technique is compared with other existing approaches of software fingerprints. In Section 3, the design of the proposed technique is described. In Section 4, the achievement of the five goals stated above is evaluated and Section 5 concludes with a summary and a description of further possible enhancements.

2 Background

Many open source toolkits such as Stigmata [2], JBirth [3] and SandMark [4] use software fingerprints to compare Java programs. With its 16 birthmark techniques, Stigmata is the most comprehensive open source birthmark toolkit available for Java.

No technique found in these toolkits can be considered equivalent to the proposed technique. The Used Classes (UC) technique (and its variants) is the closest to the proposed technique. The UC technique is based on the concept of well-known classes, which is similar to the concept of identifiable types used in the proposed technique. The UC technique relies on the list of distinct well-known classes used by a class to identify and compare classes. Compared to the proposed technique, the UC technique can be considered a coarse-grained approach. The UC technique uses class-level attributes to compare programs while the proposed technique uses method-level attributes. Moreover, the UC technique ignores types that are not considered well known such as primitive types and user-defined types. Furthermore, it does not take into account the fact that some types are used more frequently than others.

3 Design

3.1 Overview

The proposed technique is based on weighted method signatures (WMS) fingerprints. A fingerprint [22][23] is a subset of all characteristics of a program. To be effective, fingerprints must contain distinctive characteristics that are not likely to change. Method signatures possess such characteristics; thus they constitute a solid base for the proposed technique. Some method signatures are more likely to be present in a class than others. For instance, a class often contains a parameterless method that returns `void`, but seldom contains a method that returns an array of `DateFormat`. The WMS technique gives more importance to rare method signatures than to common ones. Also, the WMS technique extracts data related to method signatures from Java binary files (.class). Therefore, the technique can only be used to compare Java programs. No data is extracted from the source code, the documentation or any other resource files because programs are often distributed in binary form only.

3.2 Identifiable Types

Types are identified by their full name (i.e. their namespace and their name). For example, `java.lang.String` uniquely identifies the type String from any other Java types. Semantics-preserving code transformations can easily alter the full name of types if they are not used from outside of a program. In contrast, they cannot easily alter types contained in the Java system libraries (i.e. the primitive types and the types with a namespace that starts with `.java` or `.javax`) because they can be used by any Java programs.

The WMS technique considers the types contained in the Java system libraries as identifiable. These types can be identified by their full name. On the other hand, other types are considered unidentifiable because they are likely to be altered by semantics-preserving transformations. The WMS technique can distinguish identifiable types from unidentifiable types but cannot distinguish one unidentifiable type from another. Arrays are also identifiable; therefore, `int`, `int[]` and `int[][]` are three distinct types.

3.3 Selectivity

The WMS technique uses selectivity points to measure the weight of a type. Some types are used more frequently than others in Java. The more often a type is likely to be used in a Java program, the less its selectivity. Some applications make more intensive use of some types than others. For instance, graphical user interface applications and command line applications do not use the same types as frequently. Prior to the implementation of the technique, a statistical analysis of the type used in 192 standard eclipse plugins was carried out to overcome this problem. The analyzed plugins were assumed to be a representative sample of Java programs in general.

Selectivity points are based on this statistical analysis. The collected data demonstrated that the most infrequently used type was proportionally used 139 thousand times less frequently than the most frequently used one. If selectivity points were proportionally assigned based on the likelihood of a type to be used, the results would be very credible. However, the results would not be resilient because changing only one very infrequently used type would have too much impact on the fingerprint. On the other hand, ignoring that some types are used less frequently than others would harm the credibility of the technique. Selectivity categories solve this problem by providing a trade-off between credibility and resilience. Arbitrary decisions based on the statistical data and motivated by the need to conciliate credibility and resilience were made to define the number of selectivity categories and

the number of selectivity points assigned to each one. Selectivity categories are presented in Table 1.

Category Name	Count Per Type	Num. of Types	Total Count	Selectivity Points
Very Frequent	>= 50,000	2	230,126	1
Frequent	< 50,000	6	159,391	2
Normal	< 5,000	21	30,816	4
Infrequent	< 500	47	7,173	8
Very Infrequent	< 50	314	2,595	16
Total		390	430,101	

Table 1. Selectivity Categories

These selectivity points are the foundation of the WMS technique. It is not possible to compare fingerprints that use different selectivity categories or different selectivity points. For the sake of simplicity, the implementation of the WMS technique does not allow to change the selectivity categories.

3.4 Fingerprints

Method names, parameters names and method modifiers are not stored in the fingerprints because they are too vulnerable to semantics-preserving transformations. Only the return type, the parameters types and whether or not the method is static are taken into account. This information is then hashed for smaller memory consumption and faster comparison. Figure 1 illustrates the parsing of a method used by the class of Figure 2.

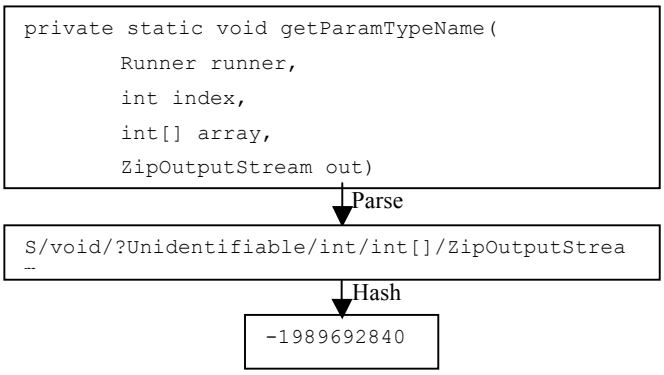


Figure 1. Method Parsing

3.5 Comparison

When comparing two JARs, the WMS technique produces a certainty percentage. This percentage represents how likely it is for the original JAR to be completely included in the compared JAR. If very strong evidence is found that 10% of the original JAR is in a copy relationship with the compared JAR, the certainty percentage will still be low because only a small portion of the original JAR was copied.

The certainty percentage is computed as follows. First, the method signature hash codes from two JARs are compared. The WMS technique does not consider similar methods; methods are either equal or different. Because the number of methods can be very large, the amount of time required for the comparison and the running time scalability of the WMS technique depend mostly on its ability to compare methods efficiently. The algorithm used to compare methods is inspired by the Merge-Join algorithm, which is widely used in relational database management system to quickly join large tables. The algorithm requires both method sets to be previously sorted by key (i.e. method signature hash code and class index). The method set is sorted when the fingerprints are created to avoid resorting for each comparison. The algorithm running time complexity is $O(n)$ (linear) if all method keys are unique and $O(n^2)$ (quadratic) if all method keys are identical. The statistical analysis performed on Eclipse plugins demonstrated that method signatures are very diverse. Therefore, comparing methods will tend to be linear.

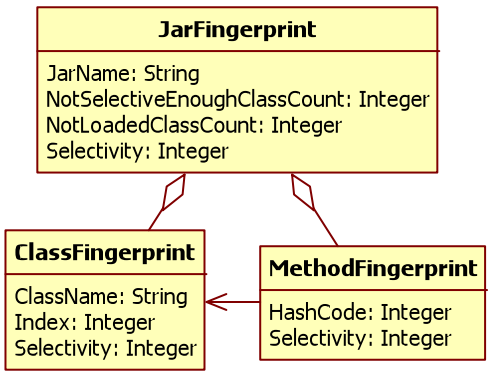


Figure 2. Fingerprint Class Diagram

The method comparison yields a matrix containing all potential matches between the original and the compared classes. Selectivity points are assigned to each potential class match. The selectivity of a class match is the sum of the selectivity of its matching methods.

To produce the certainty percentage, the next step is to identify which class matches are the best. In order to do so, class matches are sorted from the highest to the lowest selectivity. Then, the JAR match selectivity is obtained by summing up the selectivity of the best class matches. When the JAR match selectivity is computed, each original class can only be matched with one compared class and vice versa. The certainty percentage is obtained by dividing the selectivity of the JAR match by the selectivity of the original JAR.

3.6 Contextual Information

In addition to the certainty percentage, the WMS technique also provides contextual information on the copy relationship between two compared JARs. The selectivity point ratio that yielded the certainty percentage and the reverse certainty percentage are part of this contextual information. The reverse certainty percentage is an estimate of how likely is the compared JAR to be completely included in

the original JAR. This is useful when the original JAR is compared with less selective JARs.

Moreover, the name and high-level information on the JARs involved in the comparison are provided. Also the contextual information contains the 10 best class matches. The classes in this list are the ones to start with if further analysis is required to prove that copyright infringement has occurred. Also, this information is useful to detect the partial inclusion of a JAR into another one. Figure 3 exhibits the contextual information provided by the WMS technique.

3.7 Decisions Made

Low Selectivity

The statistical analysis demonstrated that 66% of classes had less than 16 selectivity points and that their impact was only 14% of the total of selectivity points. Classes with low selectivity are numerous, have a small impact on selectivity and are likely to produce false positives when compared with other classes. Therefore, excluding them produces a positive impact on each of the goals. The price to pay is that the WMS technique cannot compare JARs that are not selective enough; the WMS technique would not have yielded accurate results for these cases anyway. Out of the 192 standard eclipse plugins used for the statistical analysis, only one was considered to be not selective enough. A similar situation occurs when comparing two JARs yields matches with less than 16 selectivity points. These matches are considered to be an undesirable noise and are ignored by the comparison algorithm.

```
Certainty Percentage
    Original to Compared:                   98.5% (3802/3861)
    Compared to Original (Estimate):        96.8% (3802/3929)
Original JAR (Ori.jar)
    Selectivity:                            3861
    Number of Classes:                       237
    Not Selective Enough Classes:            167
Compared JAR (Zkm.jar)
    Selectivity:                            3929
    Number of Classes:                       237
    Not Selective Enough Classes:            167
Top 10 Class Matches
    (org.junit.Assert,gc): 100.0% (526/526)
    (junit.framework.Assert,n): 100.0% (293/293)
    (org.junit.runners.BlockJUnit4ClassRunner,ug): 100.0% (133/133)
    (junit.framework.TestSuite,cb): 100.0% (117/117)
    (junit.runner.BaseTestRunner,eb): 100.0% (117/117)
    (org.hamcrest.BaseDescription,jb): 100.0% (111/111)
    (org.junit.runner.Description,df): 94.8% (109/115)
    (org.junit.internal.runners.JUnit4ClassRunner,sg): 100.0% (104/104)
    (org.hamcrest.CoreMatchers,wb): 100.0% (95/95)
    (org.junit.runners.model.TestClass,hi): 93.1% (94/101)
```

Figure 3. WMS Technique Contextual Information

Table 2. Credibility Experiment JARs

JAR				Classes			Methods
Category	Name	Size (KB)	Selectivity	Compared	Excluded	Not Loaded	Compared
Bit Torrent Client	Vuze [5]	13,545	136,843	2,174	5,550	13	32,307
Build Script	Ant [6]	1,288	20,545	360	409	0	4,924
Code Coverage	Cobertura [7]	443	8,414	57	64	1	3,065
Code Coverage	CodeCover [8]	4,879	3,363	73	345	17	1,065
Code Coverage	EclEmma [9]	485	807	27	41	2	242
DB Test Tool	DbUnit [10]	587	8,248	175	196	14	1,306
Mp3 Player	TMp3 [11]	171	2,524	49	40	0	397
Obfuscation	Proguard [12]	658	13,502	231	310	1	3,590
Obfuscation	Sandmark [13]	5,127	44,608	817	1,103	3	10,151
Object Mocking	EasyMock [14]	109	3,463	32	46	5	481
Object Mocking	JMock [15]	235	1,486	26	48	0	246
Programming Language	Jython [16]	8,098	143,528	1,687	4,144	10	35,466
Test Sequencer	Junit [17]	238	3,833	70	167	0	717
Text Editor	JEdit [18]	3,902	44,388	677	455	0	5,929
Text Editor	JExt [19]	1,524	15,337	280	161	0	1,886

Generic Types

Generic types are problematic from the perspective of the WMS technique because they are highly selective and also vulnerable to semantics-preserving transformations. Generics are all about type safety at compile time. Therefore, replacing them with unsafe types and using the appropriate type cast at runtime would preserve the semantics of a program. To overcome this problem the generic type arguments are ignored (e.g. `List`, `List<int>` and `List<List<CustomType>>` are all equal).

Tolerance to Missing Dependencies

Java reflection is used to extract the method signature information from the Java binary files (.class). In order to load a class, Java reflection must be able to resolve all types used by this class. By convention, the WMS technique can resolve any types contained in JARs located in the same directory as the JAR from which the fingerprint is generated. If unresolvable types prevent a class from being loaded, it will be excluded from the fingerprint. A warning will notify users that some classes were not loaded.

4 Results

A series of experiments were carried out to measure if each goal was met. The experiments were executed on a machine with an Intel Core i7 920 CPU (2.65GHz), 6 GB of RAM and running Windows Vista 64 bits.

4.1 Credibility

Fifteen different JARs were used to measure the credibility of the WMS technique. Some of them accomplish similar tasks while some others were written for completely different purposes. A detailed list of the JARs used for the experiment can be found in Table 2.

In order to be credible, the WMS technique must be able to identify similar JARs without generating false positives. To measure this ability, all possible JAR combinations were compared using the WMS technique, leading to 225 comparisons (15*15).

The 15 comparisons where the JAR was compared with itself all yielded a certainty percentage of 100%. All of the 105 comparisons where the original JAR was compared with a smaller one yielded very low certainty percentage. These comparisons were not meaningful to measure credibility because the WMS technique will always yield lower certainty percentages in such cases. Therefore, these comparisons were not taken into account.

The 105 remaining comparisons did not generate any false positives. All comparisons yielded a certainty percentage under 34%, except one. The comparison between JExt and JEdit (two text editors) yielded a certainty percentage of 44.3%. However, the contextual information reported that the original and the compared class names were identical for the five best class matches. The fact that similar classes with the same name were identified provided strong evidence that the two sets of modules had not been independently written. Therefore, the result of the comparison between JExt and JEdit could not be considered a false positive.

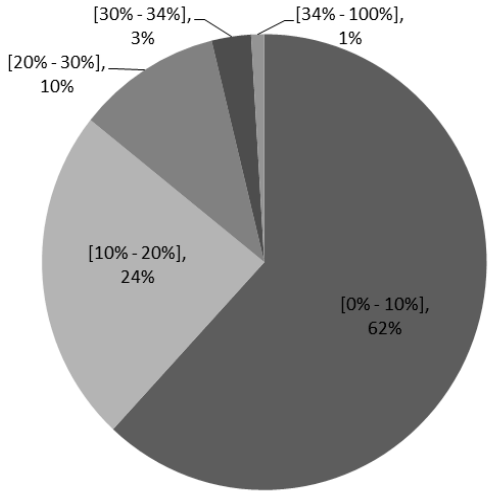


Figure 4. Certainty Percentage Distribution

4.2 Resilience to Natural Change

In order to be resilient to natural change, the WMS technique must be able to detect the copy relationship between an original work and a work derived from it. To measure this ability, seven significant releases of JUnit were compared using the WMS technique.

For each release (except one), the WMS technique was able to recognize strong similitude between the JARs. For the release of JUnit 4.0, the fact that the major digit was changed from 3 to 4 indicates that more than a minor revision was released. Also, the contextual information reported that the original and the compared class names were identical for the 10 best class matches. Moreover, each of the 10 best class matches had a certainty percentage of 100%.

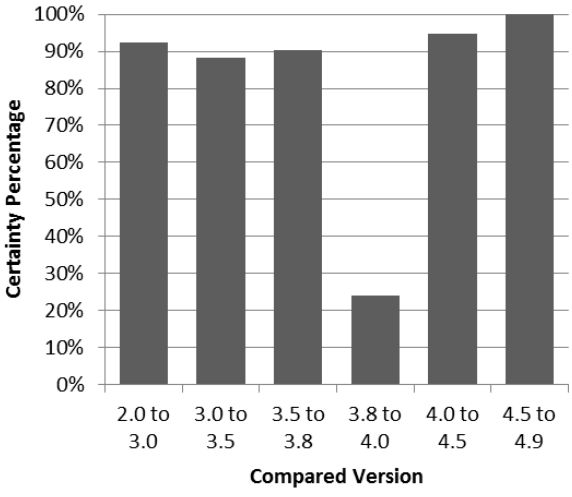


Figure 5. Copy Relationship between JUnit Releases

4.3 Resilience to Semantics-Preserving Transformations

If the WMS technique is resilient, comparing two JARs should not be affected by semantics-preserving transformations. To measure the impact of semantics-preserving transformations on the WMS technique, Pro Guard [12], Smoke Screen [20] and ZKM [21] were used to apply semantics-preserving transformations to JUnit. Then, the original JUnit JAR was compared with the obfuscated ones.

The WMS technique exhibited strong resilience to semantics-preserving transformations applied by the three obfuscators.

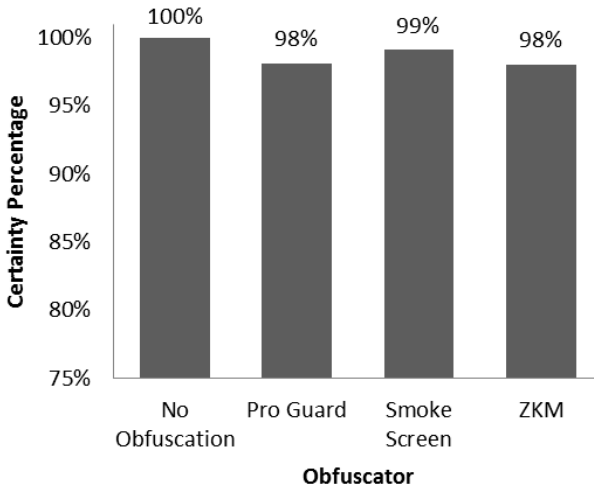


Figure 6. Obfuscation Impact

Nevertheless, being able to resist to well-known obfuscators does not mean that attacks on the WMS technique are impossible. Any semantics-preserving transformations that alter the method signatures such as reordering the parameters or promoting eligible instance methods to static would prevent the WMS technique from working properly.

Also, splitting a JAR into smaller ones or removing unused methods from the bytecode (a practice known as shrinking) would reduce the selectivity of the JAR. Such operations affect the accuracy of the certainty percentage yielded by the WMS technique. However, detecting a copy relationship in these cases would still be possible by using the reverse certainty percentage and the list of the 10 best class matches included in the contextual information.

4.4 Running Time Scalability

In order to be scalable, the time required by the WMS technique to compare two JARs must grow linearly when the size of JARs increases. Because most of the information contained in a JAR is not included in a fingerprint, the physical amount of memory required to store a JAR is not a good indicator of the size of a JAR. WMS fingerprints mostly contain information on method signatures. Thus, the number of methods contained in a JAR is a better size indicator from the perspective of the WMS technique. For the 225 comparisons performed in the credibility experiment, the relation between the number of methods to compare and the time required to perform the comparison was measured.

The results demonstrated that the average case running time complexity of the WMS technique tended to be *O(n)* (linear). Also, less than 10 seconds were required to compare the largest pair of JARs. Furthermore, the data gathered in the experiment revealed that the time required to compare two JARs was much shorter than the time required to create two fingerprints. The WMS technique could take advantage of the fast speed of the comparison when comparing multiple JARs. When the 30 fingerprints (2 sets of 15 fingerprints) were generated first and reused for all the comparisons, the WMS technique required 25 seconds to generate the fingerprints and only 12 seconds to perform the 225 comparisons.

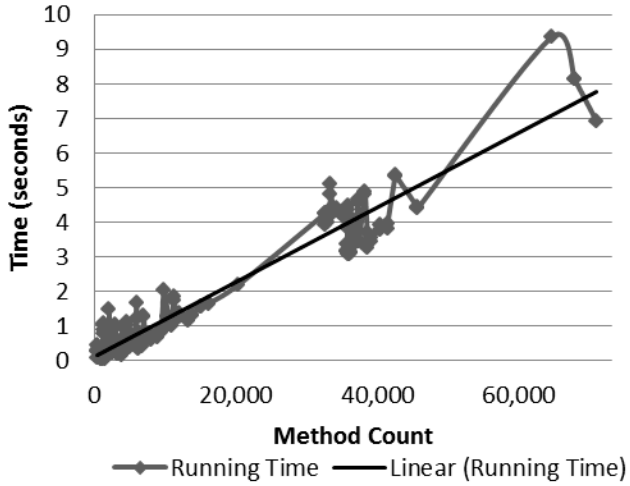


Figure 7. Running Time Scalability

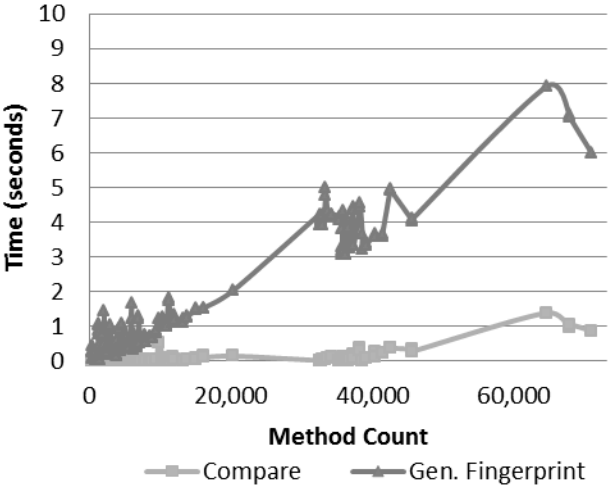


Figure 8. Running Time: Compare vs Gen. Fingerprint

4.5 Memory Usage Scalability

In order to be scalable, the size of the fingerprints must grow linearly when the size of the JARs increases. As explained in the previous experiment, the best size indicator of a JAR is the number of methods it contains. The relation between the number of methods contained in a JAR and the

size of the fingerprint was used to measure the memory usage scalability of the WMS technique.

The results confirmed that the average case memory usage complexity of the WMS technique was *O(n)* (linear). However, the sizes of large fingerprints such as Jython and Azureus were close to one megabyte.

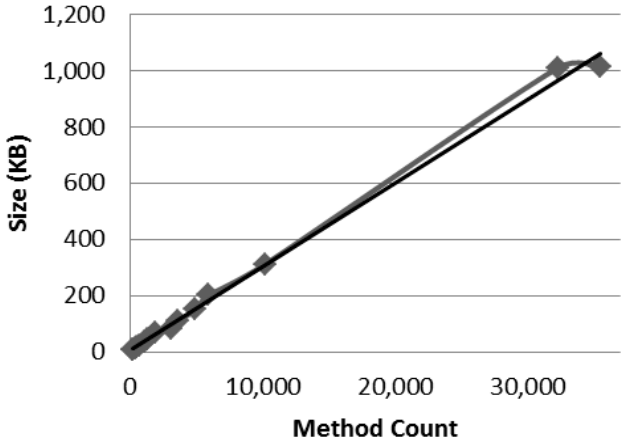


Figure 9. Fingerprint Memory Usage Scalability

5 Conclusion

A fingerprint technique that uses method signatures to compare JARs was presented in this paper. This technique recognizes that uncommon method signatures are more selective than common ones. The WMS technique cannot compare classes with low selectivity. Nevertheless, it is well suited to compare most JARs.

Figure 8 demonstrated that the generation of fingerprints is the most important performance bottleneck. A code analysis demonstrated that most of the time required to generate fingerprints is spent on parsing the method signatures using reflection. Further investigation would be required to determine if other bytecode inspection techniques could parse JAR files more efficiently.

The results of the experiments demonstrated that the WMS technique provided good trade-offs between the conflicting goals stated in the paper. The technique is cohesive enough that it could be included in a framework that allows multiple fingerprinting techniques to work in conjunction.

6 References

[1] Myles G. and Collberg C. 2005. K-gram Based Software Birthmarks. 2005 ACM Symposium on Applied Computing (March 13-17, 2005, Santa Fe, New Mexico, USA) https://mailserver.di.unipi.it/ricerca/proceedings/AppliedComputing05/PDFs/papers/T07P02.pdf

[2] Stigmata - http://stigmata.sourceforge.jp (accessed November 1, 2010)

[3] JBirth - http://se.naist.jp/jbirth (accessed November 1, 2010)

[4] SandMark - http://sandmark.cs.arizona.edu (accessed November 1, 2010)

[5] Vuze - http://www.vuze.com (accessed November 1, 2010)

[6] Ant - http://ant.apache.org (accessed November 1, 2010)

[7] Cobertura - http://cobertura.sourceforge.net (accessed November 1, 2010)

[8] CodeCover - http://codecover.org (accessed November 1, 2010)

[9] EclEmma - http://www.eclemma.org (accessed November 1, 2010)

[10] DbUnit - http://www.dbunit.org (accessed November 1, 2010)

[11] TMp3 - http://sourceforge.net/projects/tmp3 (accessed November 1, 2010)

[12] Proguard - http://proguard.sourceforge.net (accessed November 1, 2010)

[13] Sandmark - http://sandmark.cs.arizona.edu (accessed November 1, 2010)

[14] EasyMock - http://easymock.org/ (accessed November 1, 2010)

[15] JMock - http://www.jmock.org (accessed November 1, 2010)

[16] Jython - http://www.jython.org (accessed November 1, 2010)

[17] Junit - http://www.junit.org (accessed November 1, 2010)

[18] JEdit - http://www.jedit.org (accessed November 1, 2010)

[19] JExt - http://www.jext.org (accessed November 1, 2010)

[20] Smoke Screen - http://www.leesw.com/smokescreen (accessed November 1, 2010)

[21] ZKM - http://www.zelix.com (accessed November 1, 2010)

[22] Patrice Arruda, Pierre Chamoun, Dwight Deugo: A Framework for Detecting Code Piracy Using Class Structure, The 22nd International Conference on Software Engineering and Knowledge Engineering (SEKE 2010), Knowledge Systems Institute Graduate School, 559-564, 2010.

[23] Carson Brown, David Barrera, Dwight Deugo: FiGD: An Open Source Intellectual Property Violation Detector. SEKE (2009): 536-541.

Search the best greedy algorithm with Hill climbing experiments for covering array generation

Jing Jiang[1], **Changhai Nie** [1]
[1] The state key laboratory for novel software technique, Nanjing University, Nanjing, China

Abstract - *Many researchers have been working on the greedy algorithms for covering array generation; a framework has been built to integrate most of these greedy methods, and more new approaches can be derived from this framework. However, such a framework is affected by multiple dependent or independent factors, which makes its deployment and optimization very challenging. In this paper, we design Hill climbing experiments based on six decisions of the framework, aim to search the best greedy algorithm for covering array generation, the results show that the identified optimal configuration for the framework is competitive.*

Keywords: combinatorial testing, greedy methods, covering array, Hill climbing

1 Introduction

Many modern systems are built by components; they work collaboratively to enable certain system functionalities. Unexpected interactions among components may cause some potential system failure. Consequently, a good test plan should be designed to examine these interactions comprehensively. Combinatorial testing has been proposed as a means to detect failures triggered by the interactions among components in Software Under Test (SUT) [1].

Covering array generation is a key issue in combinatorial testing. Covering array is the test suite for combinatorial testing, it is an $N \times k$ array on v symbols, denoted as $MCA(N;t,k,(v_1,v_2,\cdots,v_k))$ where $v=\sum_{i=1}^{k} v_i$, t is the strength of the coverage of interactions, and k is the number of factors. Each column i ($1 \le i \le k$) contains only elements from a set V_i with $|V_i| = v_i$ .The rows of each $N \times t$ sub-array cover all t-tuples of levels from the columns at least once [1].

A greedy algorithm framework has been built by Bryce [2], many greedy methods, including AETG [3,4], TCG [5] and DDA [6,7], can be derived from it. Here we don't consider the greedy algorithm IPO [8].However, it is a new challenge that how to scientifically deploy and optimize the framework affected by multiple factors to construct covering arrays more efficiently. Bryce et al. used ANOVA to analysis the effect of each decision in array size [2], but they did not give any concrete usable configuration for producing a usable greedy algorithm. We have designed a Base Choice method to find the best configuration for the framework in our previous work[9]. In this paper, we try to design a group of experiments with Hill climbing method to address this problem. Through the experiments we can also search the best greedy algorithm. It provides theoretical and practical guideline for the design and optimization of greedy algorithms.

The remainder of this paper is organized as follows: Section 2 briefly introduces the greedy algorithm framework, Section 3 describes the Hill Climbing experiment design, Section 4 analyzes experimental data, and Section 5 presents a summary and the future work.

1. Select a factor f_i according to the factor ordering selection criterion.
2. In the case of more factors selected, then choose the factor f_i by factor tie-breaking.
3. Assign a level l_i for the factor f_i according to the level selection criterion.
4. In the case of more levels selected, then choose the level l_i by level tie-breaking.
5. Repeat the process until all factors have been fixed, then create a test case.
6. Repeat the above steps Candidates times, candidate rows are generated.
7. Choose a candidate that covers the most new pairs into the covering array.
8. Repeat the above steps until all pairs have been covered, the covering array is complete.
9. Repeat the above steps Repetitions times, then choose a smallest covering array.

Fig.1 The framework of greedy algorithms

2 Framework of greedy algorithms

Bryce et al. [2] proposed a four layer framework with six decisions from existed methods (see Figure 1). The basic idea is to build a test cast at a time, which covers as many new pairs as possible. It selects a factor in advance, and then assigns a value for the factor. This process will be repeated until all factors have been fixed. As a result, a test case is created. Figure 1 provides the detail of the greedy framework. Six decisions need to be made (see Table 1), shown in

shadows in the skeleton of process. We will explain them one by one next.

2.1 Decision one – Repetitions

Due to the randomness of certain decisions, smaller covering arrays may be generated by repetitions [2]. But the number of repetitions should not be overwhelming. If we repeat the process too many times, it may no longer reduce the array size, but bring extra cost. At this point, we only consider four kinds of repetitions: *1, 5, 10* and *20* repetitions.

2.2 Decision two – Candidates

The algorithm may generate a number of rows as candidates, and choose the one adding the most new pairs into the covering array [2]. Here we set the numbers of candidate as *1*, *5*, *10* and *20*.

2.3 Decision three -- factor ordering

The factor ordering is the essence of the framework. Researchers have refined five strategies [2]: (1) *uncovered pairs*, factors may be ranked by number of new pairs involving the factor and the fixed factors; (2) *density*, factor ordering is associated with the expected number of pairs covered involving both fixed and free factors; (3) *level*, factors are ordered by the number of associated levels; (4) *random*; (5) *hybrid factor ordering*, the first factor is selected with the most uncovered pairs left, and remaining factors are ordered randomly.

2.4 Decision four -- level selection

Its goal is to cover the largest number of new pairs. Bryce has raised three criterions [2]: (1) *random*; (2) *uncovered pairs*, by the number of new pairs involving the level of the current factor and the fixed factors; (3) *density*, by the expected number of new pairs associated with fixed and free factors.

2.5 Decision five -- factor tie-breaking

When employing the strategy of section 2.3, the algorithm may suffer from ties. To break ties, one of the following methods may be used: *take first*, *random* and *uncovered pairs*.

2.6 Decision six -- level tie-breaking

The level tie-breaking is also needed, the following methods are used: *random*, *take first*, *uncovered pairs* and *least used*.

2.7 The existing greedy methods

One possible configuration of the framework is {Repetitions = 1, Candidates =1, factor ordering = random, level selection = random, factor tie-breaking = random, level tie-breaking = random}. Each of such configurations can be deployed in the greedy framework, and form a new greedy method. Similarly the configuration is recorded as (1, 1, random, random, random, random).

The AETG [3, 4] method is represented the configuration of the framework is (-, -, hybrid, uncovered pairs, -, uncovered pairs), where the character "-"denotes that the choice can be selected arbitrarily. In AETG, the factor ordering selection is a *hybrid* rule. The method chooses the first factor as one with the most new pairs left, and the remainder is in *random* factor ordering.

When the configuration of the framework is (1, 1, density, density, take first, take first), it denotes DDA [6, 7]. Its factor ordering selection takes into account free factors. DDA selects a factor based on *density*.

The idea of TCG [5] is similar in many respects to AETG. They only have two differences: the number of candidates and factor ordering. In TCG, factors are assigned levels in a non-ascending order of each factor's cardinality, and the number of candidates is the maximum cardinality of factors.

3 Experimental designs

The issue of covering array generation is NP-Hard. Most of methods can merely get approximated solution. AETG, TCG and DDA have their own advantage on solving the specific problems, but none of them outperform each other with the respect to accuracy, efficiency, consistency and extensibility [2]. Bryce etc have constructed a greedy

Table 1: The specific strategies of six decisions

Repetitions (f_0)	Candidates (f_1)	factor ordering(f_2)	level selection(f_3)	factor tie-breaking(f_4)	Level tie-breaking(f_5)
1	1	random	random	random	random
5	5	uncovered pairs	uncovered pairs	uncovered pairs	uncovered pairs
10	10	density	density	take first	take first
20	20	level			least frequent
		hybrid			

framework with six decisions refining from the existing algorithms. Each decision has a variety of choices, and new greedy algorithms including AETG and DDA can be derived by the choice combinations of the decisions (our framework has $4\times4\times5\times3\times3\times4$=2880 methods). In this paper, we employ sampling methods to study all kinds of methods based on the framework. We focus on the performance of the generated covering array size, and aim to explore the following issues:

- Does the configuration of the framework affect the size of the generated covering array?
- If the configuration of the framework has an impact on the performance of the generated covering array size, can we find an optimal configuration to generate smaller covering arrays for some systems?
- If the optimal configuration exists in the framework, is it able to construct smaller covering arrays for most systems?
- Is the optimal configuration of the framework competitive with the existing methods as AETG, TCG and DDA?

To address these questions, we design and implement a configurable greedy algorithm tool for covering array generation (FOGM) as the experimental platform. It is able to run arbitrarily configurable greedy algorithms. The tool has two inputs: (1) SUTs, for example, the system 3445 (which means a system with 9 factors, 4 factors having 3 levels, 5 factors having 4 levels); (2) Configuration sets, the values for these six decisions in the framework, such as (5, 10, uncovered pairs, uncovered pairs, random, random).

For convenience, we use natural numbers to denote each choice of the decisions in Table 1. Suppose one decision has *n* choices, we then record the choices as *{0, 1, …,n-1}*. For example, the configuration (5, 10, uncovered pairs, uncovered pairs, random, random) is able to be indicated as the configuration (1, 2, 1, 1, 0, 0). The tool has two outputs, as the following: (1) For a special system, the best configuration of the framework and the array size; (2) the generated covering array sizes produced by each configuration.

Our experiment contains the exploration and verification of the best configuration for the greedy framework. Firstly, we produce the configuration set from Hill climbing method. With the configuration set, we systematically explore the influence of each decision, thereby find the best configuration. We then evaluate the efficiency of the best configuration, and judge whether the configuration is suitable for other systems. We also compare the configuration with the existing methods such as AETG, TCG and DDA, and examine whether it has advantages in the array size. In the first stage, we design the Hill climbing configuration, as show below:

Hill climbing configuration set： The hill climbing first selects a base configuration *C0*. Then, it begins with varying the choices of the first decision f_0 at a time, keeping the choices of the other decisions fixed in the base configuration *C0*. In this way, we can get a group of configurations. When comparing the performance of the methods based on these configurations, we select a best configuration *C1* to be the next base configuration. Then we continue changing the choices for the second decision f_1, configurations are generated based on the base configuration *C1*, repeat steps as described above until varying the sixth decision f_5 on the base configuration *C5*. The generation of the configuration set is dynamic, generated step by step in the concrete experiments. More detail can be seen in the section 4.1.

4 Experimental analysis

Using the Hill climbing configuration set, we configure the greedy algorithms, and obtain multiple greedy approaches, then generate covering arrays for the systems listed in the first column of Table 2-7, and record the *size* of the generated covering array respectively. For example, in Table 2, for the system 6^4, we can get 42 test cases in the configuration *C0*. The letter "*f*" in Table 7 denotes that the greedy method is failed to generate a covering array. The results in Table 2-7 demonstrate that the configurations of the framework have a significant impact on the performance of the generated covering array size. The experimental results are analyzed next.

4.1 Hill climbing experiments

Hill climbing is used to explore the impact on the size of the generated covering array for each decision. We determine the base configuration randomly as *C0* = (2, 2, 1, 1, 2, 0). Then we generate the configuration set step by step in the experiment. The process is as follows.

Table 2: Results for repetitions

	C0	H1	H2	H3
$5^1 3^8 2^2$	20	21	20	19
$3^4 4^5$	24	26	24	23
$6^1 5^1 4^6 3^8 2^3$	34	36	34	34
6^4	42	42	42	41
$8^2 7^2 6^2 5^2$	69	70	69	69

(1) Repetitions

Based on the configuration *C0*= (2, 2, 1, 1, 2, 0), new configurations are created by varying the number of repetitions at a time until all repetitions have been covered, keeping the other decisions fixed on *C0*. The configurations

are H1= (0, 2, 1, 1, 2, 0), H2= (1, 2, 1, 1, 2, 0) and H3= (3, 2, 1, 1, 2, 0). A configuration subset consists of the base configuration *C0* and the three configurations. The results are given in Table 2. The *repetitions* of these configurations are studied unsing settings of *10*, *1*, *5* and *20*. Comparing the results, we can draw the conclusion that more repetitions may reduce the array size, but while the repetitions are increased to some extent, the size tends to be stable, even the increase of repetitions wastes time. As the tradoff between the time and the size, so we determine the next base configuration *C1*= H3 = (3, 2, 1, 1, 2, 0) .

(2) Candidates

Repeat like the above steps, we change the value of *candidates* at a time keeping the values of the other decisions fixed on *C1*. Then the configurations will be H4= (3, 0, 1, 1, 2, 0), H5= (3, 1, 1, 1, 2, 0) and H6= (3, 3, 1, 1, 2, 0). Variations in number of *candidates* are *10*, *1*, *5* and *20* respectively. Table 3 shows that more candidates often reduce the size of the generated array size slightly, it improves the size performance at the cost of time. For example, with the configuration *C1*, the greedy algorithm can generate a covering array of 69 test cases in 27.78s, with H6, the size is 67 in 54.22s. So consider the tradeoff between the size and time cost, the next base configuration *C2* = H6 = (3, 3, 1, 1, 2, 0).

Table 3: Results for candidates

	C1	H4	H5	H6
$5^1 3^8 2^2$	19	21	20	19
$3^4 4^5$	23	25	23	23
$6^1 5^1 4^6 3^8 2^3$	34	36	34	33
6^4	42	45	42	41
$8^2 7^2 6^2 5^2$	69	82	70	67

(3) Factor ordering selection

Changing the value of *factor ordering selection* in *C2*, then the configurations are H7= (3, 3, 0, 1, 2, 0), H8= (3, 3, 2, 1, 2, 0), H9= (3, 3, 3, 1, 2, 0) and H10= (3, 3, 4, 1, 2, 0). Their choices are *uncovered pairs*, *random*, *density*, *level* and *hybrid*. The results are shown in Table 4. *Density* based factor ordering appears to be the best choice, *random* in H7 produces the worst results.Therefore the next base configuration *C3*= H8= (3, 3, 2, 1, 2, 0).

(4) Level selection

By varying the values of *level selection* in *C3*, we construct two configurations: H11= (3, 3, 2, 0, 2, 0) and H12= (3, 3, 2, 2, 2, 0). There are three *level selection* methods: *uncovered pairs*, *random* and *density*. The results are shown in Table 5. We can find that the *random* ordering has the worst performance, *uncovered pairs* and *density* obtain similar performance, both are better than *random*. Moreover, the level selection based on *uncovered pairs* is slightly better than *density*. So the next base configuration *C4*= *C3*= (3, 3, 2, 1, 2, 0).

Table 4: Results for factor ordering

	C2	H7	H8	H9	H10
$5^1 3^8 2^2$	20	20	19	20	20
$3^4 4^5$	23	24	22	23	24
$6^1 5^1 4^6 3^8 2^3$	33	38	33	33	36
6^4	41	41	41	42	42
$8^2 7^2 6^2 5^2$	67	71	66	67	70

Table 5: Results for level selection

	C3	H11	H12
$5^1 3^8 2^2$	19	26	20
$3^4 4^5$	22	31	23
$6^1 5^1 4^6 3^8 2^3$	34	51	34
6^4	41	52	41
$8^2 7^2 6^2 5^2$	67	105	69

(5) Factor tie-breaking

Changing the values of *factor tie-breaking* in the base configuration *C4*, we can get two configurations H13= (3, 3, 2, 1, 0, 0) and H14= (3, 3, 2, 1, 1, 0). Their choices are *take first*, *random* and *uncovered pairs* respectively. Results are given in Table 6. We can see that random in H13 has better performance. Therefore the next base configuration *C5*= H13 = (3, 3, 2, 1, 0, 0).

Table 6: Results for factor tie-breaking

	C4	H13	H14
$5^1 3^8 2^2$	19	19	20
$3^4 4^5$	22	22	22
$6^1 5^1 4^6 3^8 2^3$	34	33	33
6^4	41	39	41
$8^2 7^2 6^2 5^2$	67	67	68

Table 7: Results for level tie-breaking

	C5	H15	H16	H17
$5^1 3^8 2^2$	19	23	f	21
$3^4 4^5$	22	24	f	24
$6^1 5^1 4^6 3^8 2^3$	33	34	f	36
6^4	42	42	f	43
$8^2 7^2 6^2 5^2$	67	71	f	f

(6) Level tie-breaking

Varying the values of *level tie-breaking* in *C5*, we can get three configuration H15=(3,3,2,1,0,1), H16=(3,3,2,1,0,2) and H17=(3,3,2,1,0,2). The choices are *random*, *uncovered pairs*, *take first* and *least used*. As shown in Table 7, we can find that *take first* produces worst performance, frequently

suffering from a dead loop, and the algorithms using *least used* would fail occasionally. Besides the bad configurations, we can determine the next base configuration *C6* =*C5*= (3, 3, 2, 1, 0, 0).

Through the above six steps, the best configuration *C**= *C6* = (3, 3, 2, 1, 0, 0) (namely the configuration (20, 20, density, uncovered pairs, random, random)) is obtained.

4.2 Verifying experiments

To verify the optimal configuration *C**, we examine the following two aspects: 1) Is the optimal configuration able to generate smaller covering arrays for most systems; 2) Is it competitive with the existing methods as AETG, TCG and DDA? For the evaluation, we choose seven systems listed in the first column of Table 8, the results are shown in Table 8, comparing with the published results for AETG, TCG and DDA in the literature [3-7]. The first column represents the configuration *C**, and the last three columns show the existing greedy methods DDA, AETG and TCG. From Table 8, we can find the optimal configuration C* is competitive in generating covering arrays. For example, for the system $7^1 6^1 5^1 4^5 3^8 2^3$ the size by *C** is 42, the size by AETG is 45, the size by TCG is 45, and the size by DDA is 43.

Table 8: Comparison with published results

	*C**	AETG	TCG	DDA
3^{13}	18	15	*20*	18
$5^1 3^8 2^2$	19	19	20	21
$6^1 5^1 4^6 3^8 2^3$	33	34	33	34
$5^1 4^4 3^{11} 2^5$	26	30	30	27
$4^{15} 3^{17} 2^{29}$	34	41	*35*	35
$7^1 6^1 5^1 4^5 3^8 2^3$	42	45	45	43
4^{40}	43	42	*46*	43

5 Conclusions

We studied a greedy framework with six decisions built by Bryce [2]. Thousands of greedy methods can be derived from this framework. In order to evaluate the performance of these algorithms, we have designed a configurable greedy algorithm tool for covering array generation (FOGM). With this tool, we can configure and run any greedy algorithms conveniently for our research.

We designed and conducted Hill climbing experiments to systematically study the effect on the size of the generated covering array of each decision. According to the experimental results, we can draw the following conclusions: (1) the configurations of the framework have a significant impact on the performance of the covering array size. Thus, users should be careful about the selection of the decisions of the framework; (2) We can obtain an optimal configuration in some specific systems, and (3) the optimal configuration can work for the other systems as well; (4) Comparing the optimal configuration with the existing methods AETG, TCG and DDA, we find that the configuration has obvious advantages, in some systems, it is slightly better than the existing methods.

In the future work, we plan to conduct more profound and comprehensive studies on the greedy framework, which may include: (1) consider more choices of factor ordering, level selection, factor tie-breaking and level tie-breaking, such as hybrid rules based on factor or level selection; (2) employ more scientific sampling methods to optimize the configuration of the framework; (3) select more systems into the experiments; (4) consider the cases of seeds and constraints in covering array generation.

6 References

[1] Changhai Nie, Hareton Leung. A survey of combinatorial testing, ACM Computing Survey, 2011, 43(2).

[2] R. C. Bryce, C. J. Colbourn, M. B. Cohen. A Framework of Greedy Methods for Constructing Interaction Test Suites. In: Proceedings of 27th international conference on Software engineering (ICSE2005), St. Louis, Missouri, USA, May 15-21, 2005: 146--155.

[3] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The AETG system: an approach to testing based on combinatorial design. IEEE Transactions on Software Engineering, 23(7):437--44, October 1997.

[4] D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton. The combinatorial design approach to automatic test generation. IEEE Software, 13(5): 82--88, October 1996.

[5] Y. Tung and W. Aldiwan. Automating test case generation for the new generation mission software system. IEEE Aerospace Conf., pages 431--37, 2000.

[6] C. J. Colgourn, M. B. Cohen, R. C. Turban. A deterministic density algorithm for pairwise interaction coverage. Proc. of the IASTED Intl. Conference on Software Engineering, pages 242--252, Fevruary 2004.

[7] R. C. Bryce, C. J. Colbourn. The density algorithm for pairwise interaction testing. Journal of Software Testing, Verification, and Reliability, 2007.

[8] K. Tai, L. Yu. A test generation strategy for pairwise testing. IEEE Transactions on Software Engineering, 28:109--111, 2002.

[9] Jing Jiang, Changhai Nie. Find the best greedy algorithm with Base Choice experiments for covering array generation. Submit to: proceedings of 1th International Workshop on Evidential Assessment of Software Technologies (EAST2011), Beijing, China.

LR(1) Parser Generator Hyacc

X. Chen[1], **D. Pager**[1]
[1]Department of Information and Computer Science, University of Hawaii at Manoa, Honolulu, HI, USA

Abstract - *The space and time cost of LR parser generation is high. Robust and effective LR(1) parser generators are rare to find. This work employed the Knuth canonical algorithm, Pager's practical general method, lane-tracing algorithm, and other relevant algorithms, implemented an efficient, practical and open-source parser generator Hyacc in ANSI C, which supports full LR(0)/LALR(1)/LR(1) and partial LR(k), and is compatible with Yacc and Bison in input format and command line user interface. In this paper we introduce Hyacc, and give a brief overview on its architecture, parse engine, storage table, precedence and associativity handling, error handling, data structures, performance and usage.*

Keywords: Hyacc, LR(1), Parser Generator, Compiler, Software tool

1 Introduction

The canonical LR(k) algorithm [1] proposed by Knuth in 1965 is a powerful parser generation algorithm for context-free grammars. It was potentially exponential in time and space to be of practical use. Alternatives to LR(k) include the LALR(1) algorithm used in parser generators such as Yacc and later Bison, and the LL algorithm used by parser generators such as ANTLR. However, LALR and LL are not as powerful as LR. Good LR(k) parser generator remains scarce, even for the case k = 1.

This work has developed Hyacc, an efficient and practical open source full LR(0)/LALR(1)/LR(1) and partial LR(k) parser generation tool in ANSI C. It is compatible with Yacc and Bison. The LR(1) algorithms employed are based on 1) the canonical algorithm of Knuth [1], 2) the lane-tracing algorithm of Pager [2][3], which reduces parsing machine size by splitting from a LALR(1) parsing machine that contains reduce-reduce conflicts, and 3) the practical general method of Pager [4], which reduces parsing machine size by merging compatible states from a parsing machine obtained by Knuth's method. The LR(0) algorithm used in Hyacc is the traditional one. The LALR(1) algorithm used in Hyacc is based on the first phase of the lane-tracing algorithm. LR(0) and LALR(1) are implemented because Pager's lane-tracing algorithm depends on these as the first step. The LR(k) algorithm is called the edge-pushing algorithm [5] based on recursively applying the lane-tracing process, and works for a subclass of LR(k) grammars. As a side optimization, Hyacc also implemented the unit production elimination algorithm of Pager [6] and its extension [5].

2 The Hyacc Parser Generator

2.1 Overview

Hyacc is pronounced as "HiYacc". It is an efficient and practical parser generator written from scratch in ANSI C, and is easy to port to other platforms. Hyacc is open source. Version 0.9 was released in January 2008 [7]. Version 0.95 was released in April 2009. Version 0.97 was released in January 2011.

Hyacc is released under the GPL license. But the LR(1) parse engine file hyaccpar and LR(k) parse engine file hyaccpark are under the BSD license so that the parser generators created by Hyacc can be used in both open source and proprietary software. This addresses the copyright problem that Richard Stallman discussed in "Conditions for Using Bison" of his Bison manuals [8][9].

The algorithms employed by Hyacc are listed in the introduction.

Hyacc is compatible to Yacc and Bison in its input file format, ambiguous grammar handling and error handling. These directives from Yacc and Bison are implemented in Hyacc: %token, %left, %right, %expect, %start, %prec. Hyacc can be used together with the lexical analyser Lex. It can generate rich debug information in the parser generation process, and store these in a log file for review.

If specified, the generated parser can record the parsing steps in a file, which makes it easy for debugging and testing. It can also generate a Graphviz input file for the parsing machine. With this input, Graphviz can draw an image of the parsing machine.

2.2 Architecture

Hyacc first gets command line switch options, then reads from the grammar input file. Next, it creates the parsing machine according to different algorithms as specified by the command line switches. Then it writes the generated parser to y.tab.c, and optionally, y.output and y.gviz. y.tab.c is the parser with the parsing machine stored

in arrays. y.output contains all kinds of information needed by the compiler developer to understand the parser generation process and the parsing machine. y.gviz can be used as the input file to Graphviz to generate a graph of the parsing machine.

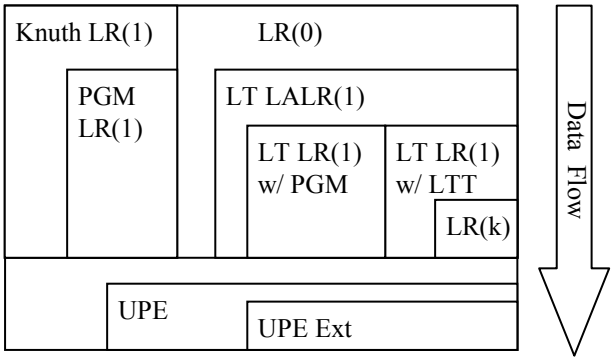


Fig. 1. Relationship of algorithms from the point of view of grammar processing [1]

Fig. 1 shows how the algorithms used in Hyacc are structured from the point of view of grammar processing. Input grammars can be processed by the merging path on the left, first by the Knuth canonical algorithm and stop here, or be further processed by Pager's PGM algorithm.

Input grammars can also be processed by the splitting path on the right. First the LR(0) parsing machine is generated. Next the LALR(1) parsing machine is generated by the first phase of the lane-tracing algorithm. If reduce-reduce conflicts exist, this is not a LALR(1) grammar, and the second phase of lane-tracing is applied to generate the LR(1) parsing machine. There are two methods for the second phase of lane-tracing. One is based on the PGM method [4], the other is based on the lane table method [10]. If LR(1) cannot resolve all the conflicts, this may be a LR(k) grammar and the LR(k) process is applied.

The generated LR(1) parsing machine may contain unit productions that can be eliminated by applying the UPE algorithm and its extension.

Fig. 2 shows the relationship of the algorithms from the point of view of implementation, i.e., how one algorithm is based on the other.

[1] Knuth LR(1) – Knuth canonical algorithm, PGM LR(1) – Pager's practical general method, LT LALR(1) – LALR(1) based on lane-tracing phase 1, LT LR(1) w/ PGM – lane-tracing LR(1) algorithm based on Pager's practical general method, LT LR(1) w/ LTT – lane-tracing LR(1) algorithm based on Pager's lane table method, UPE – Pager's unit production elimination algorithm, UPT Ext – Extension algorithm to Pager's unit production elimination algorithm.

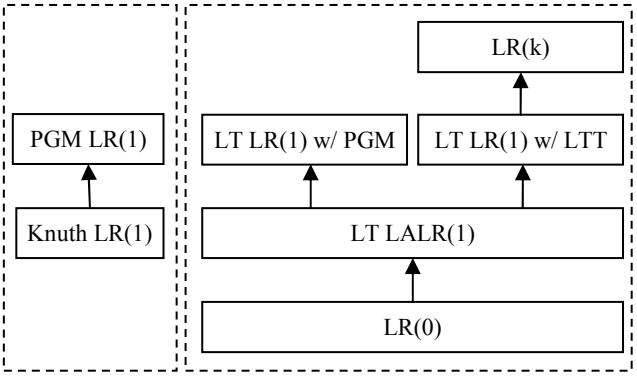


Fig. 2. Relationship of algorithms from the point of view of implementation

2.3 The LR(1) Parse Engine

Similar to the yaccpar file of Yacc, the hyaccpar file is the parse engine of Hyacc. The parser generation process embeds the parsing table into hyaccpar. How the hyaccpar LR(1) parse engine works is shown in Algorithm 1.

Algorithm 1: The hyaccpar LR(1) parse engine algorithm.[2]

```
1  Initialization:
2  push state 0 onto state_stack;

3  while next token is not EOF do {
4    S ← current state;
5    L ← next token/lookahead;
6    A ← action of(S, L) in parsing table;
7    if A is shift then {
8      push target state on state_stack,
9        pop lookahead symbol;
10     update S and L;
11   } else if A is reduce then {
12     output code for this reduction;
13     r1 ← LHS symbol of reduction A;
14     r2 ← RHS symbol count of A;
15     pop r2 states from state_stack,
16       update current state S;
17     A_tmp ← action for (S, r1);
18     push target goto state A_tmp to
         state_stack;
19   } else if A is accept then {
20     if next token is EOF then {
21       is valid accept, exit;
22     } else {
23       is error, error recovery or exit;
24     }
25   } else {
26     is error, do error recovery;
27   }
28 }
```

In Algorithm 1, a state stack is used to keep track of the current status of traversing the state machine. The parameter 'S' or current state is the state on the top of the

[2] LHS – Left Hand Side, RHS – Right Hand Side.

state stack. The parameter 'L' or lookahead is the symbol used to decide the next action from the current state. The parameter 'A' or action is the action to take, and is found by checking the parsing table entry (S, L). '←' denotes assignment operation. This parse engine is similar to the one used in Yacc, but there are variation in the details, such as the storage parsing table, as discussed in the next section.

TABLE 1. STORAGE ARRAYS FOR THE PARSING MACHINE IN HYACC PARSE ENGINE.

Array name	Explanation
yyfs[]	List the default reduction for each state. If a state has no default reduction, its entry is 0. Array size = *n*.
yyrowoffset[]	The offset of parsing table rows in arrays yyptblact[] and yyptbltok[]. Array size = *n*.
yyptblact[]	Destination state of an action (shift/goto/reduce/ accept). If yyptblact[i] is positive, action is 'shift/goto', If yyptblact[i] is negative, action is 'reduce', If yyptblact[i] is 0, action is 'accept'. If yyptblact[i] is -10000000, labels array end. Array size = *p*.
yyptbltok[]	The token for an action. If yyptbltok[i] is positive, token is terminal, If yyptbltok[i] is negative, token is non-terminal. If yyptbltok[i] is -10000001, is place holder for a row. If yyptbltok[i] is -10000000, labels array end. Array size = *p*.
yyr1[]	If the LHS symbol of rule i is a non-terminal, and its index among non-terminals (in the order of appearance in the grammar rules) is *x*, then yyr1[i] = -*x*. If the LHS symbol of rule i is a terminal and its token value is *t*, then yyr1[i] = *t*. Note yyr1[0] is a placeholder and not used. Note this is different from yyr1[] of Yacc or Bison, which only have non-terminals on the LHS of its rules, so the LHS symbol is always a non-terminal, and yyr1[i] = x, where x is defined the same as above. Array size = *r*.
yyr2[]	Same as Yacc yyr2[]. Let x[i] be the number of RHS symbols of rule i, then yyr2[i] = x[i] << 1 + y[i], where y[i] = 1 if production i has associated semantic code, y[i] = 0 otherwise. Note yyr2[0] is a placeholder and not used. This array is used to generate semantic actions. Array size = *r*.
yynts[]	List of non-terminals. This is used only in debug mode. Array size = number of non-terminals + 1.
yytoks[]	List of tokens (terminals). This is used only in debug mode. Array size = number of terminals + 1.
yyreds[]	List of the reductions. Note this does not include the augmented rule. This is used only in debug mode. Array size = *r*.

2.4 Storing the Parsing Table

2.4.1 Storage tables

The following describes the arrays that are used in hyaccpar to store the parsing table.

Let the parsing table have *n* rows (states) and m columns (number of terminals and non-terminals). Assuming there are *r* rules (including the augmented rule), and the number of non-empty entries in the parsing table is *p*. Table 1 lists all the storage arrays and explains their usage.

2.4.2 Complexity

Suppose in state i there is a token j, we can find if an action exists by looking at the yyptbltok table from yyptbltok[yyrowoffset[i]] to yyptbltok[yyrowoffset[i+1]-1]:

i) If yyptbltok[k] == j, yyptblact[k] is the associated action;
ii) If yyptblact[k] > 0, this is a 'shift/goto' action;
iii) If yyptblact[k] < 0, is a reduction, then use yyr1 and yyr2 to find number of states to pop and the next state to goto;
iv) If yyptblact[k] == 0 then it is an 'accept' action, which is valid when j is the end of an input string.

The space used by the storage is: 2n + 2p + 3r + (m + 2). In most cases the parsing table is a sparse matrix. In general, 2n + 2p + 3r + (m + 2) < n*m.

For the time used, the main factor is when searching through the yyptbltok array from yyptbltok[yyrowoffset[i]] to yyptbltok[yyrowoffset[i+1]-1]. Now it is linear search and takes O(n) time. This can be made faster by binary search, which is possible if terminals and non-terminals are sorted alphabetically. Then time complexity will be O(ln(n)). It can be made such that time complexity is O(1), by using the double displacement method which stores the entire row of each state. That would require more space though.

2.4.3 Example

An example is given to demonstrate the use of these arrays to represent the parsing table. Given grammar G1:

E → E + T | T
T → T * a | a

The parsing table is shown in Table 2. Here the parsing table has n = 8 rows, m = 6 columns, and r = 5 rules (including the augmented rule). The actual storage arrays in the hyaccpar parse engine are shown in Table 3.

Array yyfs[] lists the default reduction for each state: state 3 has default reduction on rule 4, and state 7 has default reduction on rule 3.

Array yyrowoffset[] defines the offset of parsing table rows in arrays yyptblact[] and yyptbltok[]. E.g., row 1 starts at offset 0, row 2 starts at offset 3.

Array yyptblact[] is the destination state of an action. The first entry is 97, which can be seen in the yytoks[] array. The second entry is 1, which stands for non-terminal E. And as we see in the parsing table, entry (0, a) has action s3, entry (0, E) has action g1, thus in yyptblact[] we see correspondingly the first entry is 3, and the second entry is 1. Entry -10000000 in both yyptblact[] and yyptbltok[] labels the end of the array. Entry 0 in yyptblact[] labels the accept action. Entry 0 in yyptbltok[] stands for the token end marker $. Entry -10000001 in yyptbltok[] labels that this state (row in parsing table) has no other actions but the default reduction. -10000001 is just a dummy value that is never used, and servers as a place holder so yyrowoffset[] can have a corresponding value for this row.

Entries of array yyr1[] and array yyr2[] are defined as in Table 1, and it is easy to see the correspondence of the values.

2.5 Handling Precedence and Associativity

The way that Hyacc handles precedence and associativity is the same as Yacc and Bison. By default, in a shift/reduce conflict, shift is chosen; in a reduce/reduce conflict, the reduction whose rule appears first in the grammar is chosen. But this may not be what the user wants. So %left, %right and %nonassoc are used to declare tokens and specify customized precedence and associativity.

2.6 Error Handling

Error handling is the same as in Yacc. There have been abundant complaints about the error recovery scheme of Yacc. We are concentrating on LR(1) algorithms instead of better error recovery. Also we want to keep compatible with Yacc and Bison. For these reasons we keep the way that Yacc handles errors.

2.7 Data Structures

A symbol table is implemented by hash table, and uses open-chaining to store elements in a linked list at each bucket. The symbol table is used to achieve O(1) performance for many operations. All the symbols used in the grammar are stored as a node in this symbol table. Each node also contains other information about each symbol. Such information are calculated at the time of parsing the grammar file and stored for later use.

TABLE 2. PARSING TABLE FOR GRAMMAR G1

State	*$*	*+*	***	*a*	*E*	*T*
0	0	0	0	s3	g1	g2
1	a0	s4	0	0	0	0
2	r2	r2	s5	0	0	0
3	r4	r4	r4	0	0	0
4	0	0	0	s3	0	g6
5	0	0	0	s7	0	0
6	r1	r1	s5	0	0	0
7	r3	r3	r3	0	0	0

TABLE 3. STORAGE TABLES IN HYACC LR(1) PARSE ENGINE FOR GRAMMAR G1

```
#define YYCONST const
typedef int yytabelem;

static YYCONST yytabelem yyfs[] = {0, 0,
0, -4, 0, 0, 0, -3};

static YYCONST yytabelem yyptbltok[] = {
97, -1, -2, 0, 43, 0, 43, 42, -10000001, 97,
-2, 97, 0, 43, 42, -10000001, -10000000};

static YYCONST yytabelem yyptblact[] = {
3, 1, 2, 0, 4, -2, -2, 5, -4, 3,
6, 7, -1, -1, 5, -3, -10000000};

static YYCONST yytabelem yyrowoffset[] = {
0, 3, 5, 8, 9, 11, 12, 15, 16};

static YYCONST yytabelem yyr1[] = {
     0,    -1,    -1,    -2,    -2};
static YYCONST yytabelem yyr2[] = {
     0,     6,     2,     6,     2};

#ifdef YYDEBUG

typedef struct {char *t_name; int t_val;}
yytoktype;

yytoktype yynts[] = {
  "E", -1,
  "T", -2,
  "-unknown-", 1  /* ends search */
};
yytoktype yytoks[] = {
  "a", 97,
  "+", 43,
  "*", 42,
  "-unknown-", -1  /* ends search */
};
char * yyreds[] = {
  "-no such reduction-"
  "E : 'E' '+' 'T'",
  "E : 'T'",
  "T : 'T' '*' 'a'",
  "T : 'a'",
};
#endif /* YYDEBUG */
```

Linked list, static arrays, dynamic arrays and hash tables are used where appropriate. Sometimes multiple data structures are used for the same object, and which one to use depends on the particular circumstance.

In the parsing table, the rows index the states (e.g., row 1 represents actions of state 1), and the columns stand for the lookahead symbols (both terminals and non-terminals) upon which shift/goto/reduce/accept actions take place. The parsing table is implemented as a one dimensional integer array. Each entry [row, col] is accessed as entry [row * column size + col]. In the parsing table positive numbers are for 'shift', negative numbers are for 'reduce', -10000000 is for 'accept' and 0 is for 'error'.

There is no size limit for any data structures. They can grow until they consume all the memory. However, Hyacc artificially sets an upper limit of 512 characters for the maximal length of a symbol.

2.8 Performance

The performance of Hyacc is compared to other LR(1) parser generators. Menhir [11] and MSTA [12] are both very efficiently and robustly implemented. Table 4 and Table 5 show the running time comparison of the three parser generators on C++ and C grammars. The speeds are similar. MSTA, implemented in C++, is a handy choice for industry users. It does not use reduced-space LR(1) algorithms though, thus always results in larger parsing machines. Menhir uses Pager's PGM algorithm, but is implemented in Caml, which is not so popular in industry. Therefore Hyacc should be a favorable choice.

TABLE 4. RUNNING TIME (SEC) COMPARISON OF MENHIR, MSTA AND HYACC ON C++ GRAMMAR.

	Knuth LR(1)	PG MLR(1)	LALR(1)
Menhir	1.97	1.48	N/A
MSTA	5.32	N/A	1.17
Hyacc	3.53	1.78	1.10

TABLE 5. RUNNING TIME (SEC) COMPARISON OF MENHIR, MSTA AND HYACC ON C GRAMMAR.

	Knuth LR(1)	PG MLR(1)	LALR(1)
Menhir	1.64	0.56	N/A
MSTA	0.92	N/A	0.13
Hyacc	1.05	0.42	0.19

2.9 Usage

From the sourceforge.net homepage of Hyacc [7] a user can download the source packages for unix/linux and windows, and the binary package for windows. All the instructions on installation and usage are available in the included readme file. It is very easy to use, especially for users familiar with Yacc and/or Bison.

Hyacc is a command line utility. To start hyacc, use: "hyacc input_file.y [-bcCdDghKlmnoOPQRStvV]". The input grammar file input_file.y has the same format as those used by Yacc/Bison.

The meanings of some of the command line switches are briefly introduced here. '-b' specifies the prefix to use for all hyacc output file names. The default is y.tab.c as in Yacc. If '-c' is specified, no parser files (y.tab.c and y.tab.h) will be generated. This is used when the user only wants to use the -v and -D options to parse the grammar and check the y.output log file. '-D' is used with a number from 0 to 15 (e.g., -D7) to specify the details to be included into the y.output log file during the parser generation process. '-g' says that a Graphviz input file should be generated. '-S' means to apply LR(0) algorithm. '-R' applies LALR(1) algorithm. '-Oi' where i = 0 to 3 applies the Knuth canonical algorithm and the practical general method with different optimizations. '-P' applies the lane-tracing algorithm based on the practical general method. '-Q' applies the lane-tracing algorithm based on the lane table method. '-K' applies the LR(k) algorithm. '-m' shows man page.

For more usage of the Hyacc parser generator, interested users can refer to the Hyacc usage manual.

3 Related Work

Pager's practical general method has been implemented in some other parser generators. Some examples are LR (1979, in Fortran 66, at Lawrence Livermore National Laboratory) [13], LRSYS (1985, in Pascal, at Lawrence Livermore National Laboratory) [14], LALR (1988, in MACRO-11 under RSX-11) [15], Menhir (2004, in Caml) [11] and the Python Parsing module (2007, in Python) [16].

The lane-tracing algorithm was implemented by Pager (1970s, in Assembly under OS 360) [3]. But no other available implementation is known.

For other LR(1) parser generators, the Muskox parser generator (1994) [17] implemented Spector's LR(1) algorithm [18][19]. MSTA (2002) [12] took a splitting approach but the detail is unknown. Commercial products include Yacc++ (LR(1) was added around 1990, using splitting approach loosely based on Spector's algorithm) [20][21] and Dr. Parse (detail unknown) [22]. Most recently an IELR(1) algorithm [23][24] was proposed to provide LR(1) solution to non-LR(1) grammars with specifications to solve conflicts, and the authors implemented this as an extension of Bison.

4 Conclusions

In this work we investigated LR(1) parser generation algorithms and implemented a parser generator Hyacc, which supports LR(0)/LALR(1)/LR(1) and partial LR(k). It has been released to the open source community. The usage of Hyacc is highly similar to the widely used LALR(1) parser generators Yacc and Bison, which makes it easy to learn and use. Hyacc is unique in its wide span of algorithms coverage, efficiency, portability, usability and availability.

5 References

[1] Donald E. Knuth. On the translation of languages from left to right. Information and Control, 8(6):607 – 639, 1965.

[2] David Pager. The lane tracing algorithm for constructing LR(k) parsers. In Proceedings of the fifth annual ACM symposium on Theory of computing, pages 172 – 181, Austin, Texas, United States, 1973.

[3] David Pager. The lane-tracing algorithm for constructing LR(k) parsers and ways of enhancing its efficiency. Information Sciences, 12:19 – 42,

[4] David Pager. A practical general method for constructing LR(k) parsers. Acta Informatica, 7:249 – 268, 1977.

[5] Xin Chen. Measuring and Extending LR(1) Parser Generation. PhD thesis, University of Hawaii, August 2009.

[6] David Pager. Eliminating unit productions from LR parsers. Acta Informatics, 9:31 – 59, 1977.

[7] Xin Chen. LR(1) Parser Generator Hyacc, January 2008. http://hyacc.sourceforge.net.

[8] Charles Donnelly, Richard Stallman. Bison, The YACC-compatible Parser generator (for Bison Version 1.23), 1993.

[9] Charles Donnelly, Richard Stallman. Bison, The YACC-compatible Parser generator (for Bison Version 1.24), 1995.

[10] David Pager. The Lane Table Method Of Constructing LR(1) Parsers. Technical Report No. ICS2009-06-02, University of Hawaii, Information and Computer Sciences Department, 2008. http://www.ics.hawaii.edu/research/tech-reports/LaneTableMethod.pdf/view

[11] Francois Pottier and Yann Regis-Gianas. Parser Generator Menhir, 2004. http://cristal.inria.fr/~fpottier/menhir/

[12] Vladimir Makarov. Toolset COCOM & scripting language DINO, 2002. http://sourceforge.net/projects/cocom

[13] Charles Wetherell and A. Shannon. LR automatic parser generator and LR(1) parser. Technical Report UCRL-82926 Preprint, July 1979.

[14] LRSYS, 1991. http://www.nea.fr/abs/html/nesc9721.html

[15] Algirdas Pakstas, 1992. http://compilers.iecc.com/comparch/article/92-08-109

[16] Parser Generator Parsing.py: An LR(1) parser generator with CFSM/GLR drivers, 2007. http://compilers.iecc.com/comparch/article/07-03-076

[17] Boris Burshteyn. MUSKOX Algorithms, 1994. http://compilers.iecc.com/comparch/article/94-03-067

[18] David Spector. Full LR(1) parser generation. ACM SIGPLAN Notices, p.58 – 66, 1981.

[19] David Spector. Efficient full LR(1) parser generation. ACM SIGPLAN Notices, 23(12):143 – 150, 1988.

[20] Yacc++ and the Language Objects Library, 1997 – 2004. http://world.std.com/~compres

[21] Chris Clark, 2005. http://compilers.iecc.com/comparch/article/05-06-124

[22] Parser Generator Dr. Parse. http://www.downloadatoz.com/software-development_directory/dr-parse

[23] Joel E. Denny, Brian A. Malloy. IELR(1): practical LR(1) parser tables for non-LR(1) grammars with conflict resolution. Proceedings of the 2008 ACM symposium on Applied computing, p.240 – 245, 2008.

[24] Joel E. Denny, Brian A. Malloy, The IELR(1) algorithm for generating minimal LR(1) parser tables for non-LR(1) grammars with conflict resolution, Science of Computer Programming, v.75 n.11, p.943 – 979, November, 2010.

Lightweight Formal Verification for Tail Recursive Loops

A. Ricardo Morales and J. Nelson Rushton
Department of Computer Science, Texas Tech University, Lubbock, Texas, U.S.A.

Abstract—*A formal method is given for generating a correctness argument from commented code for a tail recursive function. The generated argument is a set of propositions which, if true, guarantee the partial correctness of the code with respect to its documented specification (where by partial correctness, we mean that if the function returns anything at all then it returns the correct value). The intended use of the system is to teach students to write loop invariants.*

Keywords: Software Engineering, Program Verification, Functional Programming, Programming Pedagogy.

1. Introduction

This paper presents a formal method for generating a correctness argument from commented code for a tail recursive function. The method is formal in the sense that there is an algorithm for generating a set F of formulas from the commented code, such that if the propositions of F are true then the code is partially correct (that is, cannot return an incorrect value, though termination is not necessarily guaranteed). However, the checking of the propositions of F is left to intuition. Thus it might be said that the algorithm generates a proof, which is correct if the code is correctly written and commented — but that checking the proof is up to the programmer.

The *real* purpose of the method is to help teach students to comment code with preconditions, postconditions, and invariants. In order to teach invariants we need not only a precise definition of invariant, but also a precise definition of a "key" invariant that can be used in a correctness argument — for if we simply require students to write an invariant for a loop that fits the textbook definition, then $0 = 0$ will always do.

To make matters more challenging, we would like to give a criterion for key invariants that does not depend on the notion of a full formal argument. That is, we would like to define the class of invariants that *could* be used in a proof, independently of the notion of the proof itself. This allows invariants to be taught precisely, in contexts where there is not enough room in the syllabus to fully develop material on formal arguments.

For example, Code Sample 1 can be proven correct using the invariant $a = i!$ (in the absence of overflow errors, which will be addressed in Section 3). However, if we simply ask students to state *an* invariant of the tail recursive loop, then $0 = 0$ is is also a correct answer. What we seek is a precise definition that distinguishes key invariants from useless ones, without reference to their use in a full proof.

Code Sample 1:

```
; function fac
; precondition: n is a natural number
; postcondition: (fac n) = n!

(define (fac n) (facit 0 1 n))

(define (facit i a n) (cond
   [(= i n) a]
   [else (facit (add1 i) (* a (add1 i)) n)]))
```

The material presented here was used in a course on the theory of programming languages. By itself it took one day of class time. To understand it, students are required to have a basic familiarity with of formal logical propositions and functional programming, but *not* necessarily with formal inference rules or proofs. The most important learning outcome, in our view, is that students gain an intuition for writing invariants for tail recursive loops. The derivation of a correctness argument is simply a formalization of what *we* do intuitively in our heads, to convince ourselves that we have written an appropriate invariant.

2. Tail recursion

Code for a tail recursive function in Racket (a freely available dialect of Lisp) may be written as follows:

(`define` $(f\ a_1 \ldots a_n\ )($ `define` $(\ h\ S_1 \ldots S_m\ ))$
(`define`$(\ h\ x_1 \ldots x_m\ )($`cond` $[\ G_1\ B_1\ ] \ldots [\ G_k\ B_k\ ]))$

where f, h, all a_i, and all x_i are symbols, and all S_i, G_i, and B_i are expressions. Each B_i must either be a base case which contains no recursive calls to h (or any function defined in terms of h), or a tail recursive case of the form $(h\ E_{i,1} \ldots E_{i,m}))$, where the $E_{i,j}$'s contain no calls to h (or any function which depends on h).

As a constraint on the use of this method, we assume that if f and h have any parameters in common, then these parameters are passed by h to itself unchanged in each recursive call. More precisely, if $x_i = a_j$ for some i and j, then for each recursive call $(h\ E_{k,1} \ldots E_{k,m})$ we have that $E_{k,j}$ is the symbol a_j.

3. Invariants

Good documentation for the tail recursive function shown in Code Sample 1 should also include an invariant. Loosely speaking, the invariant is a property of the arguments of the recursive helper h which has the following three properties:

1) it is true when h is first called,
2) it remains true for each recursive call to h, and
3) it guarantees a correct return value.

For example, our factorial function could be documented as follows:

$$;\texttt{invariant}:\texttt{a} = \texttt{i}!$$

In evaluating the expression (`fac 3`), for example, the following calls to `facit` will be generated, in order:

(`facit 0 1 3`)
(`facit 1 1 3`)
(`facit 2 2 3`)
(`facit 3 6 3`)

The invariant says that for each call (`facit` i a n) made during evaluation, we have $a = i!$. Note this is true in the trace above. Since execution stops only when $i = n$, this means we have $a = n!$ when execution halts. Since the expression returned is a, this implies that $n!$ is returned, as desired.

Next we will describe more precisely the properties which the invariant must have in relation to the pre- and post-conditions. The precondition is stated as a formula PRE whose free variables are among $a_1 \ldots a_n$. The postcondition is stated as a formula $POST$ whose free variables are also among $a_1 \ldots a_n$. The invariant is stated as a sentence INV whose free variables are among $x_1 \ldots x_m$.

Condition 1, that the invariant must hold on the first call to h, is written formally as

$$PRE \Rightarrow INV(S_1, \ldots, S_m) \quad (1)$$

where $INV(S_1, \ldots, S_m)$ is the sentence obtained from INV by substituting S_j for x_j, $1 \leq j \leq m$.

Condition 2, that the invariant remains true in each recursive call, corresponds, for each recursive case $[G_i\ (h\ E_{i,1} \ldots E_{i,m})]$ to a sentence of the form

$$INV \wedge \neg(G_1) \wedge \ldots \wedge \neg(G_{i-1}) \wedge G_i \Rightarrow INV(E_{i,1},\ \ldots,\ E_{i,m}) \quad (2)$$

where $INV(E_{i,1},\ \ldots,\ E_{i,m})$ is the sentence obtained from INV by substituting $E_{i,j}$ for x_j , $1 \leq j \leq m$.

Condition 3, that the invariant guarantees a correct return value corresponds, for each base case $[G_i\ B_i]$, to a sentence of the form

$$INV \wedge \neg(G_1) \wedge \ldots \wedge \neg(G_{i-1}) \wedge G_i \Rightarrow POST(B_i) \quad (3)$$

where $POST(B_i)$ is the sentence obtained from $POST$ by substituting B_i for $(f\ a_1 \ldots a_n)$.

These three formulas can be considered as premises for a valid argument that the defined function never returns an incorrect value. That is, if the formulas are true in the intended interpretation, then the function cannot returns an incorrect value.

In the case of our factorial function in Code Sample 1, conditions (1), (2), and (3) are instantiated as follows:

$$n \in \mathbb{N} \wedge n \geq 0 \quad \Rightarrow \quad 0 \neq 1 \quad (4)$$

$$a = i! \wedge \neg(i = n) \quad \Rightarrow \quad (* \ a\ (\texttt{add1}\ i)) = (\texttt{add1}\ i)! \quad (5)$$

$$a = i! \wedge i = n \quad \Rightarrow \quad a = n! \quad (6)$$

Unfortunately, statement (5) is not true. As a skeptical reader may have guessed before now, the supposed invariant fails to hold if execution of a recursive call results in an arithmetic overflow. One way to handle this is to write a new precondition and invariant that guarantee there is no arithmetic overflow. Racket has big integers built in, and it can store integers up to 1000! (and, in fact, much higher). Thus a fully documented factorial implementation can be written as follows:

Code Sample 2:

```
; function fac
; precondition: n in N  and 0<=n<=1000
; postcondition: (fac n) = n!

(define (fac n) (facit 0 1 n))

; invariant: 0<=i<=n<=1000 and a=i!

(define (facit i a n) (cond
    [(= i n) a]
    [else (facit (add1 i) (* a (add1 i)) n)]))
```

The corresponding (correct) premises are as follows:

$$n \in \mathbb{N} \wedge 0 \leq n \leq 1000 \quad \Rightarrow \quad 0! = 1 \quad (7)$$

$$0 \leq i \leq n \leq 1000 \wedge a = i! \wedge \neg(i = n) \Rightarrow 0 \leq (\texttt{add1}\ i) \leq n \leq 1000 \wedge (* \ a\ (\texttt{add1}\ i)) = (\texttt{add1}\ i)! \quad (8)$$

$$0 \leq i \leq n \leq 1000 \wedge a = i! \wedge i = n \quad \Rightarrow \quad a = n! \quad (9)$$

In a more general context, these three steps would be the base case, induction step, and final step of a a proof by induction that the function is partially correct (that is, cannot return an incorrect value). The induction proof can be automatically generated from these three key steps, and is correct if premises (1)-(3) are sound. This gives a method of proving that a tail recursive function never returns an incorrect value, which requires only three lines of documentation (at least two of which, and arguably all of which, ought to be written anyway), and a tool to generate the key proof steps so that they may be examined during a code review.

4. Another Example

Here is another simple example:

Code Sample 3:

```
;maximum
;precondition: L is a nonempty list of numbers
;postcondition: (maximum L)
;                  is the largest element of L

(define (maximum L) (maxit (car L) (cdr L)))

;invariant: MAXIMUM(L) = MAX(x,MAXIMUM(s))

(define (maxit x s) (cond
   [(empty? s) x]
   [(< x (car s)) (maxit  (car s) (cdr s))]
   [else (maxit x (cdr s))]))
```

For built-in functions, we will adopt the convention that an identifier in all caps denotes the mathematical function implemented by the corresponding operator in lower case letters. For user defined functions the convention is that the all-caps identifier denotes an ideal implementation satisfying $PRE \Rightarrow POST$. For example, here *MAXIMUM* is the mathematical function which is *supposed* to be implemented by the Racket function with the same name in lower case. This convention was found to be helpful because (1) the desired behavior will always need to be referred to in the argument for correctness, and (2) there is often no formal specification which is practically more clear than our pre-existing intuitive picture of the desired behavior. Some students seemed to think this was circular, but the following explanation helped: *MAXIMUM* is the function we want to write; $maximum$ is the function we actually wrote; and the question is whether they are the same. That question cannot be answered without talking about both.

The premises for the correctness argument for Code Sample 3 are as follows:

$$\begin{gathered}(L \text{ is a nonempty list of numbers}) \\ \Rightarrow \\ \textit{MAXIMUM}(L) = MAX((\texttt{car}\ L), \textit{MAXIMUM}(\texttt{cdr}\ L))\end{gathered} \tag{10}$$

$$\begin{gathered}\textit{MAXIMUM}(L) = MAX(x, \textit{MAXIMUM}(s)) \wedge \\ \neg(\texttt{empty?}\ s) \wedge x < (\texttt{car}\ s) \Rightarrow \\ \textit{MAXIMUM}(L) = MAX((\texttt{car}\ s), \textit{MAXIMUM}(\texttt{cdr}\ s))\end{gathered} \tag{11}$$

$$\begin{gathered}\textit{MAXIMUM}(L) = MAX(x, \textit{MAXIMUM}(s)) \wedge \\ \neg(\texttt{empty?}\ s) \wedge \neg\ x < (\texttt{car}\ s) \Rightarrow \\ \textit{MAXIMUM}(L) = MAX(x, \textit{MAXIMUM}(\texttt{cdr}\ s))\end{gathered} \tag{12}$$

$$\begin{gathered}\textit{MAXIMUM}(L) = MAX(x, \textit{MAXIMUM}(s)) \wedge \\ (\texttt{empty?}\ s) \Rightarrow \\ \textit{MAXIMUM}(L) = x\end{gathered} \tag{13}$$

One can check that these four statements are true, and infer from this the partial correctness of the code.

5. Conclusions and Future Work

This material was covered in one lecture of an hour and twenty minutes, in the junior level course *Concepts of Programming Languages* at Texas Tech, during Fall 2010. Four questions were given on the final exam which covered the material, requiring students to write tail recursive solutions for finding Fibonacci numbers and reversing a list, document their code with variants and invariants, and write formal premises corresponding to the preservation of truth of the invariant, and the guarantee of a correct return value. Out of 52 students who took the exam, 9 displayed mastery on these questions (as measured by receiving at least 90% credit on them) and 19 displayed either competency or mastery (by receiving at least 75% credit). This was a class where most students displayed competency or mastery on most material; and so these results seem to indicate a problem with either the material or the presentation. It should be noted that the course, of which this material was a major component, was well received in general, ranking higher than any other course in the department for Fall 2010 as measured by student responses to the question *was this course effective overall?*

We propose a two part hypothesis to explain the data:

1) Like all methods of rigorous or formal reasoning, the approach requires mathematical sophistication that must be built up over a period of years, and is generally not exercised by other classes in our curriculum.
2) The examples used to demonstrate the method were all of different sorts, requiring different mathematical concepts to invent an appropriate invariant.

Item (2) seemed like a good idea at the time. Indeed it is certainly beneficial to work several kinds of examples. However, having *no two examples alike* meant that students were loaded with novel concepts at every turn *in addition* to the target material.

We hypothesize that this can be improved by repeating several examples using the same design pattern (e.g., stepping through successive integers from low to high) before moving to another design pattern. Additionally, we hypothesize that the method itself can be streamlined in most cases by stating special case theorems that apply to common design patterns, such as recursively popping through a list or stepping through consecutive integers. This would be analogous to stating the Pythagorean Theorem and using it for the common special case of right triangles, instead of using the more general (and more complex) law of cosines. In the next iteration we plan to augment the theory with simplified methods for important special cases and demonstrate several uses of each.

Formal Modeling of Navigation System of Autonomous Mobile Robots using Graphs, Automata and Z

Nazir Ahmad Zafar[1], **Fahad Alhumaidan**[1] **and Javed Iqbal**[2]
[1] Department of Computer Science, King Faisal University, Saudi Arabia
Emails: {nazafar, falhumaidan}@kfu.edu.sa
[2] Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan
Email: javedsamon@ucp.edu.pk

Abstract - *In recent years, interest has been developed for the problem of mobile robot navigation system because of its applications in various disciplines. The software development of mobile robot navigation is a complex phenomena and it is passed through different stages of design. It can be modeled either for autonomous, usually, in an unknown environment or for controlled robots in a known environment. Finite automata and graph theory have proved to be useful tools in modeling the robot navigation system through discrete environment, for example, building with rooms, doors, stairs and passageways. In this research, we have used an integration of graph theory, automata and Z notation. In finite automata, an automatic movement of robot is described in which the states are represented by nodes (rooms) and transitions (passageways, stairs and doors) by directed edges. In the mobile robot navigation system, a robot travels from a start state to a specified final state. Graphs are used to model and analyze the paths from start state to the final state. Using Z/Eves, it is investigated and analyzed the entire formal specification.*

Keywords: Navigation System; Integration of Approaches; Formal Methods; Graph Theory; Automata and Z

1 Introduction

Applications of mobile robots are visible in various disciplines, for example, manufacturing, construction, waste management, undersea work, space exploration, medical surgery, serving and assistance for the disabled. Mobile robots navigation has a broad domain covering a large spectrum of applications and technologies. It draws on some ancient as well as the advanced techniques. Mobile robot navigation system can be modeled either for autonomous in an unknown environment or for controlled robots in a known environment. Making progress toward autonomous robots is of major practical interest in a wide variety of applications.

The ability to navigate from one location to another is a fundamental capability of any mobile robot. Avoiding the dangerous situation such as collision with obstacles and staying in the safe operational environment comes first but navigation is also required. Navigation is an ability to identify its own position and then navigate towards the destination. To achieve the quality of navigation, different approaches have been proposed. In this research, an integrated approach is applied to model mobile robot navigation in which first we examine the problem of assigning a heterogeneous population of rescue robots to reach a set of target rooms where a set of people may be trapped. The mobile robot navigation system has attracted a much attention of many researchers and different navigation methodologies have been proposed. A model to guide and keep track of a robot such that it is able to complete several tasks is described in [1]. In other works, behavior based navigation is developed using dynamic systems theory to generate the behaviors [2]. Petri-nets are used as a modeling tool for the navigation in unstructured environments [3]. Now the existing navigation of single robot has extended to multiple robots system. The navigation of multiple robots addressing the issues of cooperation and formation control is discussed in [4] and [5] where reactive behavior based approach to formation control is described. Some other similar work can be found in [6-13].

The work carried out by the Melo, Isabel and Lima in [14], [15] has examined the problem of multi-robot navigation. They have analyzed the problem of driving a robot population moving in a discrete environment from some initial to a target configuration. Finite automata and graph theory have proved to be useful mathematical models for robot navigation through a discrete environment [16], [17]. In [18], finite automata are used to control the navigation of mobile robot along the possible paths. They analyzed the movement and controlled the population of a mobile robot by applying the algorithms of graph theory. A number of modeling techniques for mobile robot navigation have been developed by researchers such as partially observable Markov and behavior based navigation but the existing approaches have lack of the formalization. In this paper, the integration of graphs and automata is used for modeling of mobile robot navigation system. Z notation is used for formal description and Z/Eves tool is used for its analyses. Rest of the paper is organized as follow.

In sections 1 and 2, introduction to automata theory and formal methods is given. In section 3, formal modeling of mobile navigation system using integration of approaches is described. Finally, conclusion and future work are discussed in section 4.

2 Automata and Formal Methods

Automata are theoretical models of computer known as abstract machines which are deterministic or deterministic and used for modeling finite state systems. A finite state automaton represents a problem as a series of states with transitions between the states caused by an input to it. The automaton starts from an initial state and an input causes a transition from the current to the next state which may be an accepting state [19]. A deterministic finite automaton (DFA) consists of a finite set of states denoted Q, finite set of input symbols denoted $\sum$, transition function that takes in its argument a state and an input symbol and returns the same or a new state. The transition function is commonly denoted by δ. The start state is denoted by q0 which is one of the states in Q. The set of final states is denoted by F. The navigation automaton is a generalization of the DFA in which the alphabet is replaced by an event set and an extra active event set is introduced. The navigation automaton can be defined by a six-tuple, $G = (Y, E, f, \Gamma, Y_0, Y_m)$ where (i) Y is the state space, (ii) E is the set of possible events, (iii) $f : D \rightarrow Y$, $D \subseteq Y \times E$ are the transition functions, (iv) $\Gamma: Y \rightarrow 2^E$ is the active event function, (v) Y_0 is the initial state and (vi) Y_m is the set of marked or accepting states or also called the target states.
Formal methods based on mathematical notations are used for designing, specification and verification of software systems. Formal methods can be used at any stage of development and can be used for removing the flaws at early stages of system development. If such errors are discovered at later stages of a system, it will result costly treatment [20]. Further, correctness of a system can also be verified by using formal method [21]. Graphical notations or natural languages are the basis of traditional approaches which are used to write the specification and make the specifications highly ambiguous [22]. Formal specification based on mathematical notations has same interpretation throughout the world [23]. Every software system implicitly uses a theorem that if some condition are satisfied or not to accomplish the requirement of a software system thus there is need of code verification and further analysis. Formal methods are useful at all of these stages, i.e., requirements analysis, model checking, theorem proving and code verification [24].

3 Automata and Formal Methods

The design of mobile robot navigation system requires its environment, behavior and other functionality. Environment is modeled using graphs. Automaton is used for modeling control behavior and operations are defined in the Z notation.

3.1 Specifying Connectivity

To formalize the connectivity in Z notation, node and edge are denoted by Node and Edge as an abstract data type respectively. Each element in the set of nodes is of type Node therefore nodes is a power set of Node. To describe the set of edges, variable edges, of type power set of Edge is introduced. A walk from any start node to any target node is denoted by a schema Connectivity.

$[Node]; Edge == Node \times Node$

Connectivity

$nodes: \mathbb{P}\ Node$
$edges: \mathbb{P}\ Edge$

$nodes \neq \{\}$
$\forall e: Edge \mid e \in edges \bullet \exists n1, n2: Node \mid n1 \in nodes \land n2 \in nodes \bullet e = (n1, n2)$
$\forall n1, n2: Node \mid n1 \in nodes \land n2 \in nodes$
$\bullet \exists w: \text{seq } Edge; walk: \text{seq } Node \mid \text{ran } w \subseteq edges \land \text{ran } walk \subseteq nodes \bullet \forall i: \mathbb{Z}; e: Edge \mid i \in \text{dom } walk \land i + 1 \in \text{dom } walk \land 1 \leqslant i \leqslant \# walk \land i+1 \leqslant \# walk \land walk\ 1 = n1 \land walk\ (\# walk) = n2 \bullet (walk\ i, walk\ (i + 1)) \in \text{ran } w$

Invariants: (i) The set of nodes is a nonempty set. (ii) For each edge e of type Edge in set edges, there exists two nodes n1 and n2 belong to set of nodes and (n1, n2) equal to e which is an edge. (iii) For each start node n1 and the final node n2 there is sequence of edges such that first node is n1 and the final node is n2 and each order pair (walk i, (walk i+1)) which is an edge belongs to the range of walk.

3.2 Specification of Directed Graph

The difference between connectivity and the directed graph is that the element E of graph is now a set of order pairs of directed edges. If an edge is a directed from n1 to n2 then it is written as (n1, n2) and the opposite direction is recorded as (n2, n1). In schema *DirectedGraph,* the Ξ*Connectivity* indicates that *Connectivity* state is not changed.

DirectedGraph

$\Xi\ Connectivity$

$\forall n1, n2: Node \mid n1 \in nodes \land n2 \in nodes$
$\bullet ((n1, n2) \in edges \Rightarrow n1 \neq n2) \land ((n2, n1) \in edges \Rightarrow n1 \neq n2)$
$\lor ((n1, n2) \in edges \Rightarrow n1 \neq n2) \land ((n2, n1) \notin edges \Rightarrow n1 \neq n2)$
$\lor ((n1, n2) \notin edges \Rightarrow n1 \neq n2) \land ((n2, n1) \in edges \Rightarrow n1 \neq n2)$
$\lor ((n1, n2) \notin edges \Rightarrow n1 \neq n2) \land ((n2, n1) \notin edges \Rightarrow n1 \neq n2)$

Invariants: (i) For all nodes n1 and n2 of type Node which belong to a set of nodes where n1 and n2 are two different, there exists an edge from n1 to n2 which implies that there may not exist an edge from n2 to n1. It is to be noted that the order pairs (n1, n2) and (n2, n1) are different elements.

3.3 Specification of Environment

In the schema given below, the rooms are represented by nodes and doors, passageways and stairs are represented as directed edges. To differentiate the edges we introduce *EdgeTypes* which contains different set of edges where each set can represent the doors, passageways and stairs.

For all set of edges s that belong to the *EdgeTypes*, for each s1 and s2 of type *Edge* in set s, there exists n and n2 of type *Node* that belong to a set of nodes. And there is an edge from n1 to n2 and an edge from n2 to n1 in the set edges where s1 is equal to (n, n1) and s2 is equal to (n1, n). All set s of type power set of *Edge* where s in a set *EdgeTypes* are the subset of edges. All sets s and s1 of type power set of *Edge* in a set *EdgeTypes*, the intersection of s and s1 is equal to ∅ therefore s and s1 are distinct.

Environment

$\Xi DirectedGraph$
$EdgeTypes: \mathbb{P}\ (\mathbb{P}\ Edge)$

$\forall s: \mathbb{P}\ Edge \mid s \in EdgeTypes$
$\bullet\ \forall s1, s2: Edge \mid s1 \in s \land s2 \in s$
$\bullet\ \exists n, n1: Node \mid n \in nodes \land n1 \in nodes$
$\bullet\ ((n, n1) \in edges \land (n1, n) \in edges \Rightarrow s1 = (n, n1) \land s2 = (n1, n))$
$\lor\ ((n, n1) \in edges \land (n1, n) \notin edges \Rightarrow s1 = (n, n1) \land s2 \neq (n1, n))$
$\lor\ ((n, n1) \notin edges \land (n1, n) \in edges \Rightarrow s1 = (n1, n) \land s2 \neq (n, n1))$
$\forall s: \mathbb{P}\ Edge \mid s \in EdgeTypes \bullet s \subseteq edges$
$\forall s, s1: \mathbb{P}\ Edge \mid s \in EdgeTypes$
$\land\ s1 \in EdgeTypes \bullet s \cap s1 = \emptyset \Rightarrow s \neq s1$

3.4 Formalization of Allowed Movements

The allowed movement of robots are given below by comparing the, characteristics, properties and environment.

AMovementofEachRobot

$\Xi Environment$
$properties: \mathbb{P}\ P;\ robots: \mathbb{P}\ Robot$
$probot: Robot \rightarrow \mathbb{P}\ P$
$amovement: Robot \rightarrow \mathbb{P}\ Edge$

$\forall r: Robot \mid r \in robots \bullet probot\ r \subseteq properties$
$robots = \text{dom}\ probot$
$\forall r: Robot; p: \mathbb{P}\ P \mid r \in robots \land p \subseteq properties \bullet (r, p) \in probot$
$robots = \text{dom}\ amovement$
$\forall r: Robot; n, n1: Node \mid r \in robots \land n \in nodes \land n1 \in nodes$
$\bullet\ (n, n1) \in amovement\ r$
$\forall r: Robot; e: Edge; s: \mathbb{P}\ Edge \mid r \in robots \land s \subseteq edges \land e \in s$
$\bullet\ (r, s) \in amovement$
$\forall r: Robot \mid r \in \text{dom}\ probot$
$\bullet \forall n: Node; s: \mathbb{P}\ Edge; e, e1: Edge \mid n \in nodes \land s \in EdgeTypes$
$\bullet\ \forall n2: Node \mid n2 \in nodes \bullet \forall p: P; e: Edge \mid p \in probot\ r$
$\bullet\ ((n, n2) \in s \land (n2, n) \notin s \Rightarrow e = (n, n2) \land e \in amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \notin s \Rightarrow e = (n, n2) \land e \notin amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \notin s \Rightarrow e = (n2, n) \land e \in amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \notin s \Rightarrow e = (n2, n) \land e \notin amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \in s \Rightarrow e = (n, n2) \land e \in amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \in s \Rightarrow e = (n2, n) \land e \in amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \in s \Rightarrow e = (n, n2) \land e \notin amovement\ r)$
$\lor\ ((n, n2) \in s \land (n2, n) \in s \Rightarrow e = (n2, n) \land e \notin amovement\ r)$

In the formal specification of allowed movement of each robot, the specification of *Environment* is reused. In the schema *AMovementofEachRobot*, a new abstract data types *P* and *Robot* are introduced which define the properties of robot. In this schema a variable *properties* is a set of type power set of *P*. To define different types of robots, variable *robots* of type power set of *Robot* is introduced. The function *probot* of type Robot → ℙ (P) is used to describe the properties of each robot that there must be a unique set of type P. The function *amovement* of type *Robot* → ℙ (Edge) is introduced to describe the allowed movement of each robot. It is also described that there must be a unique set of type ℙ Edge.

All robots *r* of type *Robot* in a set of robots acting on function *probot* provide a set of properties which is subset of *properties* and domain of *probot* which is equal to a set of robots. For all robots *r* of type *Robot* in a set of *robots* and p of type power set of *P* a subset of properties where domain of *amovement* is equal to *robots* and (r, p) is in function *probot*. For each robot r of type *Robot* in a set of robots, n1 and n2 belong to a set of nodes and (n, n1) in a function *amovement* by acting on r. All robot r of type *Robot* in a set of robots and e of type *Edge* a subset of edges, where *e* is in set s and s is of power set of *Edge*, and (r, s) is in function *amovement*.

3.5 Specification of Automata

In the formal definition of automata, five components are assumed. The first one is a set of states, second is a set of alphabets, third one is the transition function, next is initial state and last one is the set of final states. Each element q in set of *astates* is type Q therefore *astates* is a type of power set of Q. To describe the set of alphabets, variable *alphabets* of type power set of X is introduced. The transition function *atrans* of type Q × X → Q is defined to describe the transition of the automata. For each input (q, x) where q is a state and x is an alphabet there must be a unique output q1 of type Q which is produced after action of *atrans* on (q, x) .

Automata

$astates: \mathbb{P}\ Q;\ alphabets: \mathbb{P}\ X$
$atrans: Q \times X \rightarrow Q;\ q0: Q;\ qf: \mathbb{P}\ Q$

$astates \neq \{\} \land q0 \in astates \land qf \subseteq astates$
$\forall q: Q; x: X \mid q \in astates \land x \in alphabets$
$\bullet\ \exists q1: Q \mid q1 \in astates \bullet atrans\ (q, x) = q1$

Invariants: (i) The set of states *astates* is nonempty. (ii) The *q0* is a start state. (iii) The set of final states *qf* is a subset of states. (iv) For each input alphabet x and states q and q1, the (atrans (q, x) , q1) describes the transition function.

3.6 Specification of Navigation Automata

In the schema *NavigationAutomata*, Y is a set of states and E is a set of events. Each element y in set states is of type Y therefore states is a type of power set of Y. To describe the set of events, variable *events* of type power set of E is

defined. The transition function f of type $Y \times E \rightarrow Y$ is introduced to describe the transition of the navigation automata, for each input (y, e) where y is a state and e is an event there must be a unique output y1 of type Y. The function *possibleaction* of type $Y \rightarrow \mathbb{P}\ E$ is introduced to describe the active events at a particular state. In which for each input y where y is a state there must be a unique output of type power set of Y.

NavigationAutomata

$states: \mathbb{P}\ Y$; $events: \mathbb{P}\ E$
$f: Y \times E \rightarrow Y$
$possibleaction: Y \rightarrow \mathbb{P}\ E$
$y0: Y$; $Ym: \mathbb{P}\ Y$

$states \neq \{\} \land y0 \in states \land Ym \subseteq states$
$\forall y: Y; e: E \mid y \in states \land e \in events$
$\bullet\ \exists y1: Y \mid y1 \in states \bullet f(y, e) = y1$
$\forall y: Y \mid y \in states \bullet \exists\ e: \mathbb{P}\ E \mid e \subseteq events \bullet possibleaction\ y = e$

3.7 Graph Navigation Automata

In this section, the environment and navigation automata in terms of graph navigation automata are described. In the navigation automata, the events and states are replaced by an edge set and a node set for the environment respectively. In the schema *GraphNAutomata*, the abstract data types *Node* and *Edge* are used. We reused the specification of *environment* without state change. Each element y in set of states is of type *Node* therefore states is a type of power set of *Node*. To describe the set of events, variable events of type power set of *Edge* is defined. The transition function f of type $f: Node \times Edge \rightarrow Node$ is introduced to describe the transition of the graph navigation automata. For each input (y, e) where y is a state and e is an event there must be a unique output y1 of type *Node*. The *possibleaction* function is of type $Node \rightarrow \mathbb{P}\ Edge$ which is introduced to describe the active events at a particular state. In this function, for each input y where y is a state there must be a unique output of type power set of *Node*.

GraphNAutomata

$\Xi Environment$
$states: \mathbb{P}\ Node; events: \mathbb{P}\ Edge$
$f: Node \times Edge \rightarrow Node$
$possibleaction: Node \rightarrow \mathbb{P}\ Edge$
$y0: Node; Ym: \mathbb{P}\ Node$

$states = nodes \land events = edges$
$states \neq \{\} \land events \neq \{\}$
$y0 \in states \land Ym \subseteq states$
$\forall y: Node; e: Edge \mid y \in states \land e \in events$
$\bullet\ \exists y1: Node \mid y1 \in states \bullet (y, e) \in \text{dom}\ f \land y1 \in \text{ran}\ f$
$\forall y: Y \mid y \in states \bullet \exists\ e: \mathbb{P}\ Edge \mid e \subseteq events \bullet possibleaction\ y = e$

The set of states is equal to nodes and set of events is equal to edges. The sets states and events are both nonempty set. The element *y0* is a start state and the set of marked states *Ym* is subset of states. For each input event e and states y and y1, f describes the transition function acting (y, e).

3.8 Graph Navigation Automata for all Robots

In the schema *ΔGraphNAutomata*, it is described that the state of *GraphNAutomata* is changed. The input variable robot? of type *Robot* is also introduced. In the schema it is described that the *robot?* is in the set of robots and the *events'* is equal to the allowed movement of *robot?*.

GraphNAutomataForAllRobot

$\Xi AMovementofEachRobot$
$\Delta GraphNAutomata$
$robot?: Robot$

$robot? \in robots$
$events' = amovement\ robot?$

3.9 Description of Path

A walk in a graph is a finite sequence of alternating nodes and edges or it can be defined by only finite sequence of edges. A path is a finite sequence of edges from starting node to the final node with distinct nodes. The variable walk is introduced to define the sequence of nodes from n1? to n2?, where n2? is the final node. Moreover, n1? and n2? are input variables of type *Node* which is used to check that either a walk exits in between n1? and n2?.

Path

$\Xi\ DirectedGraph$
$walk:$ seq $Node; n1?: Node; n2?: Node$

$\text{ran}\ walk \subseteq nodes$
$n1? \in nodes \land n2? \in nodes$
$\exists w:$ seq $Edge \mid \text{ran}\ w \subseteq edges \bullet \forall i: \mathbb{Z}; e: Edge$
$\mid i \in \text{dom}\ walk \land 1 \leqslant i \leqslant \#\ walk \land i + 1 \in \text{dom}\ walk \land walk\ 1 =$
$n1? \land walk\ (\#\ walk) = n2? \bullet (walk\ i, walk\ (i + 1)) \in \text{ran}\ w$
$\forall i, j: \mathbb{Z}$
$\mid i \in \text{dom}\ walk \land j \in \text{dom}\ walk \land 1 \leqslant i \leqslant \#\ walk \land i+1 \leqslant \#$
$walk \land 1 \leqslant j \leqslant \#\ walk \land i \neq j \bullet walk\ i \neq walk\ j$

The range of walk is a subset of nodes. The n1? and n2? are elements of nodes. There exists w of type sequence of *Edge*, which is subset of edges, and for all i in the domain of walk where the value of i is greater than or equals to 1 and less than or equal to the cardinality of walk, walk 1 is equal to n1? and the cardinality of walk is equal to n2? and (walk i, walk (i + 1)) is an edge in the range of w. For all i and j of type integers in the domain of walk, where the values of i and j are greater than 0 and less than or equal to cardinality of walk, and $i \neq j$ implies that walk $i \neq$ walk j.

3.10 Successful Path of Robot

To verify the behavior of graph navigation automata for all robots, we get a sequence of input edges and check if there exists a successful path for that particular sequence of input. In schema *SuccessfulPahtofRobot*, we reused the specification of Ξ*GraphNAutomataForAllRobot* and Ξ*Path*. The input variable navigate? is of type of sequence of edges.

SuccessfulPathofRobot

Ξ*GraphNAutomataForAllRobot*
Ξ*Path; navigate?:* seq *Edge*

---

ran *navigate?* ⊆ *events*
∀*q: Node; i:* ℤ*; pa:* ℙ *Edge*
| 1 ⩽ *i* ⩽ *# navigate?* ∧ *q* ∈ *states* ∧ *pa* ⊆ *events'*
∧ *i* ∈ dom *walk* ∧ *i* + 1 ∈ dom *walk*
• *i* =1∧*q*= *y0* ⇒(∃*q1: Node* |*q1*∈ *statesz*
•(((*q, navigate? i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *navigate? i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*))
∧ 1 < *i* < *# navigate?* ⇒ (∃*q1: Node* | *q1* ∈ *states*
• (((*q, navigate? i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *navigate? i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*))
∧ *i* = *# navigate?* ⇒ (∃*q1: Node* | *q1* ∈ *Ym*
• ((*q, navigate? i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *navigate? i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*)

The range of *navigate?* is the subset of *events*. For all q of type *Node* in set of states, *pa* of type *ℙ Edge* and i of type ℤ where i is greater than or equal to 1 and less than or equal to the cardinality of navigate?, i and i+1 belong to domain of walk. If i is equal to 1 and q is equal to the y0 then there exists q1 of type *Node* in the set of states, (q, navigate? i), q1) which belongs to function f, and (q, pa) belongs to function *possibleaction* and navigate? i belongs to the pa, walk i is equal to q and walk (i + 1) is equal to the q1.If i is greater than 1 and less than cardinality navigate? then there exists q1 in the set of *states*, (q, navigate? i), q1) belong to function f , and (q, pa) belong to function *possibleaction* then navigate? i belong to the pa , walk i is equal to q and walk (i + 1) is equal to the q1. If the value of i is equal to the cardinality of navigate? then (q, navigate? i), q1) belong to function f, and (q, pa) belong to function *possibleaction*.

3.11 All Successful Paths of Robot

In the schema *AllSuccessfulPahtofRobot,* we have reused the specification of schema Ξ*GraphNAutomataForAllRobot* and Ξ*Path*. The Ξ*Path* means that the state of schema *Path* is not changed but only reused. Similar is the definition of schema Ξ*GraphNAutomataForAllRobot*. The input variable *navigates?* is of type of power set of seq *Edge*.

For each n of type of seq Edge in the set of navigates? and for all q of type the Node in set of states, pa of type ℙ Edge is the subset of events and i of type ℤ, where i is greater than or equal to 1 and less than or equal to the cardinality of n, also i and i+1 belong to domain of walk. When the value of i is equal to 1 and q is equal to the y0 then there exists q1 of type Node in the set of states, (q, n i), q1) belong to the function f, and (q, pa) belong to function *possibleaction* then n i belong to the pa , walk i is equal to q and walk (i + 1) is equal to the q1.When the value of i is greater than 1 and less than cardinality i then there exists q1 of type Node in the set of states, (q, n i), q1) belong to the function f , and (q, pa) belong to function possibleaction then n i belong to the pa , walk i is equal to q and walk (i + 1) is equal to the q1. When the value of i is equal to the cardinality of n then there exists q1 of type Node in the set of final states Ym, (q, n i), q1) belong to the function f, and (q, pa) belong to function *possibleaction* then n i belong to the pa, walk i is equal to q and walk (i + 1) is equal to the q1.

AllSuccessfulPathofRobot

Ξ*GraphNAutomataForAllRobot*
Ξ*Path; navigates?:* ℙ (seq *Edge*)

∀*n:* seq *Edge* | *n* ∈ *navigates?*
• ∀*q: Node; i:* ℤ*; pa:* ℙ *Edge* | 1 ⩽ *i* ⩽ *# n*∧ *i*+1 ⩽ *# n* ∧ *q* ∈ *states* ∧ *pa* ⊆ *events*
∧ *i* ∈ dom *walk* ∧ *i* + 1 ∈ dom *walk*
• *i* = 1 ∧ *q* = *y0* ⇒ (∃*q1: Node* | *q1* ∈ *states*
• (((*q, n i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *n i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*))
∧ 1 < *i* < *# n* ⇒ (∃*q1: Node* | *q1* ∈ *states*
• (((*q, n i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *n i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*))
∧ *i* = *# n* ⇒ (∃*q1: Node* | *q1* ∈ *Ym*
• ((*q, n i*), *q1*) ∈ *f* ∧ (*q, pa*) ∈ *possibleaction*
⇒ *n i* ∈ *pa* ∧ *walk i* = *q* ∧ *walk* (*i* + 1) = *q1*)

---

4 Conclusions

The problem of autonomous robots navigation system is currently an intensively studied research area in the domain of intelligent and autonomous systems. In this paper, we have presented an integrated approach for modeling of mobile robot navigation system using graph theory, automata and Z notation. Automata and graph theory have proved to be useful tools in modeling the robot navigation system. Using automata, an automatic movement of robot is described in which the rooms are assumed as states and passageways, stairs and doors are taken as transitions. It is further assumed that a robot travels from a start state to a specified final state by a sequence of connected edges. Graph being a special case of automata is used to model and analyze the paths from start state to the final states. The Z/Eves tool is used to investigate and validate the formal models produced by describing mobile robot navigation system using above integration. The population of mobile robots have different properties to perform action in the environment, the movements of each

robot is modeled accordingly. Allowed movement of each robot was modeled by the graph navigation automata for all robots then successful path to navigate from a specified start state to a set of target states was described.

As we know that an appropriate and meaningful description is a fundamental requirement to design and develop a complex system. For this purpose, notations with well-defined syntax and semantics are required. In addition to it, a single approach is not sufficient for the description of a complete system. That is why the problem under hand the mobile robot navigation system in a known environment is modeled by integrating graph theory, navigation automata and Z notation. This integrated approach supported us to model the robot environment, its control behavior and functionality by capturing both its static and dynamics parts from an abstract to a detailed level of specification.

5 References

[1] R. Simmons and S. Koenig, "Probabilistic robot navigation in partially observable environments," Proceedings of the International Joint Conference on Artificial Intelligence, pp. 1080–1087, 1995.

[2] A.Steinhage, "Dynamical systems for the generation of navigation behaviour," PhD dissertation, Germany, 1997.

[3] R. D. Hale, et al. "Control of mobile robots in unstructured environments using discrete event modeling," SPIE International Symposium on Intelligent Systems and Advanced Manufacturing, 1999.

[4] T. Balch and M. Hybinette, "Social potentials for scalable multi robot formations," IEEE International Conference on Robotics and Automation, pp. 73–80, 2000.

[5] T. Balch and R. C.Arkin, "Behavior-based formation control for multi-robot teams," IEEE Transactions on Robotics and Automation, vol. 14(6), pp. 926-939, 1998.

[6] O. Hachour, "Path planning of autonomous mobile robot," International Journal of Systems Applications, Engineering & Development, vol. 2(4), 2008.

[7] O. Hachour and N. Mastorakis, "Avoiding obstacles using FPGA: a new solution and application," 5th International Conference on Automation & Information (ICAI), vol. 3(9), pp2827-2834, 2004.

[8] O. Hachour and N. Mastorakis, "Behaviour of intelligent autonomous ROBOTIC IAR," IASME transaction, vol. 1(1), pp. 76-86, 2004.

[9] O. Hachour and N. Mastorakism, "Intelligent Control and planning of IAR," 3rd International Multiconfrence on System Science and Engineering, 2004.

[10] B. P. Gerkey, M. J. Mataric, "Principled communication for dynamic multi-robot task allocation," Experimental Robotics Springer Verlag, Heidelberg, pp.353-362, 2001.

[11] S. Saripalli, G. S. Sukhatme and J. F. Montgomery, "An experimental study of the autonomous helicopter landing problem," the Eight International Symposium on Experimental Robotics, pp. 8-11, 2002.

[12] D. Estrin, D. Culler, K. Pister and G. Sukhatme, "Connecting the phsyical world with pervasive networks," IEEE Pervasive Computing, pp. 59-69, 2002

[13] T. Willeke, C. Kunz, I. Nourbakhsh, "The personal rover project: the comprehensive design of a domestic personal robot," Robotics and Autonomous Systems, Elsevier Science, pp.245-258, 2003.

[14] F. A. Melo, M. R. Isabel and P. Lima, "Event-driven modeling and control of a mobile robot population," Proceedings of the 8th Conference on Intelligent Autonomous Systems, pp. 237-244, 2004.

[15] J. Canny, "The complexity of robot motion planning," Cambridge, MIT Press, 1988.

[16] A. Biran, and M. Breiner, "MATLAB for engineers," Addison-Wesley Longman Publishing, 1997.

[17] J. L Gross and J. Yellen, "Handbook of graph theory," 2nd Edition, CRC Press, 2005.

[18] N. A. Zafar, N. Sabir and A. Ali, "Construction of intersection of nondeterministic finite automata using Z notation," WASET, 2008, vol. 30, pp. 96-101.

[19] J. E., Hopcropft, R. Motwani and J. D. Ullman, "Introduction to automata theory, language and computation," Addison-Wesley, 2001.

[20] J. M. Wing, "A specifier's introduction to formal methods," IEEE Computer, vol. 23(9), pp. 8-24, 1990.

[21] J. Seung-Ju, et al., "Design of software security verification with formal method tools," International Journal of Computer and Network Security, vol. 6, 2006.

[22] R. S. Pressman, "Software engineering a practitioner's approach," 5th Edition, McGraw-Hill, 2001.

[23] J. P. Bowen and M. G. Hinchey, "The use of industrial-strength formal methods," 21st International Computer Software & Application Conference, pp.332-337, 1997.

[24] V. George and R. Vaughn, "Application of lightweight formal methods in requirement engineering," The Journal of Defense Software Engineering, 2003.

COMPARATIVE STUDY OF SOFTWARE COMPLEXITIES OF TREE SEARCH ALGORITHMS

Salako R.J. and Aremu D.R.

Department of Computer Science, University of Ilorin, Ilorin, Nigeria.

Abstract - *This research paper studies the complexities of four tree search algorithms in order to determine the most efficient programming language for implementing each of the algorithms. Each the tree search algorithm was implemented in C, C++, Pascal, and Visual BASIC programming languages. The codes were empirically analysed using Halstead Volume and Cyclomatic number. The result of the analysis revealed that Pascal programming language is the best language for implementing breadth-first, depth-first, and depth-limited search algorithms while C language was isolated as the best for implementing A-Star search algorithm.*

Keywords: Tree search, implementation, complexity metrics, software complexity

1. Introduction

Given two or more software that solve a particular problem, a programmer is faced with the problem of choice of the most efficient one in terms of quantitative measure of quality, understanding, difficulty of testing and maintenance, as well as the measure of ease of using the software. The analysis of algorithm is a major task in computing. A computer scientist, most especially a programmer, who is faced with the problem of choosing an appropriate algorithm to solve his problem from myriad of available ones, may have his problem solved by analyzing the complexity of each of these algorithms in order to know the most efficient one. This is a nontrivial issue that leads to the analysis of algorithms and the means by which they can be compared. The aim of this paper is to study the complexities of four tree search algorithms in order to determine the most efficient programming language for implementing each of the algorithms. The objective to achieve this aim was to study tree search algorithms such as breadth-first, depth-first, and depth-limited, and to implement each search algorithm in C C++, Pascal, and Visual Basic programming languages. We analysed the codes of each of the algorithms empirically using Halstead Volume and Cyclomatic number. The result of the analysis showed that Pascal programming language is the best language for implementing breadth-first, depth-first, and depth-limited search algorithms, while C language was best for implementing A-Star search algorithm.

The rest of the paper is organized as follows: Section 2 presented the related work, while section 3 discussed complexity measurement. In section 4, we discussed the tree search algorithms. Section 5 presented the results of the complexity measurements, while section 6 concluded the paper.

2. Related Work

Algorithms are frequently assessed by the execution time, memory demand, and by the accuracy or optimality of the results. For practical use, another important aspect is the implementation complex. An algorithm which is complex to implement required skilled developers, longer implementation time, and has a higher risk of implementation errors. Moreover, complicated algorithms tend to be highly specialized and they do

not necessarily work well when the problem changes Akkanen. et al, (2000).

Algorithm analysis is an important part of a broader computational complexity theory, which provides theoretical estimate for the resources needed by any algorithm which solve a given computational problem. These estimates provide an insight into reasonable direction of search of efficient algorithms Jimmy Waks, (2000).

Algorithm can be studied theoretically or empirically. Theoretical analysis allows mathematical proofs of the execution time of algorithms which studies how an algorithm behaves with typical inputs. It is therefore tend to focus on the execution time and optimality of the result Sedgewick, (1995). Complexities of tree search algorithms have been mostly evaluated either mathematically or by computing the computer execution time. Neither of the two approaches is good enough for practical and realistic purpose especially in the situation where more than one algorithm exists for solving a given problem or class of problems. There is a need therefore to seek for pragmatic means of computing complexity of algorithms. Empirical analysis focuses on the implementation complexity by using software complexity measures available. In the realm of software metrics, code is looked at as output of labour. The complexity of a piece of software is thought of in the same way as the complexity of an automobile; the number of parts and the nature of the assembly may affect the amount of labour and time needed to create the end product.

Parse And Oman, (1995) applied a maintenance metrics index to measure the maintainability of C source code before and after maintenance activities. This technique allows the project engineers to track health of the code as it was being maintained. Maintainability is accessed but not in term of risk assessment.

Stark, (1996) collected and analyzed metrics in the categories of customer satisfaction, cost, and schedule with the objective of focusing management's attention on improvement areas and tracking improvements over time. This approach aided management in deciding whether to include changes in the current release, with possible schedule slippage, or include the changes in the next release. However, the author did not relate these metrics to risk assessment.

Okeyinka, (2003) designed a scan machine (software) that could evaluate the complexity of computer programs written in pascal language. The tool designed can be used to identify the most efficient algorithm from among myriad of algorithms solving the same problem.

3. Complexity Measurement

Complexity of an algorithm is the determination of the amount of resources such as time and storage necessary to develop, maintain, and execute the algorithm. Other items to be considered under resources are: (a) Man-hours needed to supervise, comprehend code, test, maintain, and change software. (b) Travel expenses, (c) The amount of re-used code modules, (d) Secretarial and technical support, etc. In this section, we presented a brief review of algorithm complexities measurement.

3.1 HALSTEAD Complexity Measure

Halstead complexity measure was developed to measure a program module's complexity directly from source code, with emphasis on computational complexity. The measures were developed by the late Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module (Halstead, 1977). The Halstead measures are based on four scalar numbers derived directly from a program's source code. n_1 = the number of distinct operators, n_2 = the number of distinct operands

N_1 = the total number of operators, N_2 = total number of operands

Table 1 : Complexity Measurement

Measure	Symbol	Formula
Program Length	N	$N = N_1 + N_2$
Program Vocabulary	N	$n = n_1 + n_2$
Program Volume	V	$V = N*(LOG_2n)$
Program Difficulty	D	$D = (n_1/_2)*(N_2/n_2)$
Program Effort	E	$E = D*V$

3.2 Cyclomatic Complexity Measure

Cyclomatic complexity directly measures the number of linearly independent paths through a program's source code. Cyclomatic complexity is computed using a graph that describes the control flow of the program. The nodes of the graph correspond to the program. A directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity, v(G), is derived from a flow graph and is mathematically computed using graph theory. More simply stated, it is found by determining the number of decision statements in a program: v (G) = e – n + p

v(G) is a cyclomatic complexity.

e is the number of edges in the flow graph

n is the number of nodes in the flow graph, and p is the connected components.

4. Tree Search Algorithms

In this section, we presented tree search algorithms for consideration of algorithm complexity measurement.

4.1 Breadth-first Search

A tree-search in which the adjacency lists of the vertices of *T* are considered on a first-come first-served basis, that is, in increasing order of their time of incorporation into *T*, is known as *breadth-first search*. In order to implement this algorithm efficiently, vertices in the tree are kept in a *queue*; this is just a list *Q* which is updated either by adding a new element to one end (the *tail* of *Q*) or removing an element from the other end (the *head* of *Q*). At any moment, the queue *Q* comprises all vertices from which the current tree could potentially be grown. Initially, at time $t = 0$, the queue *Q* is empty. Whenever a new vertex is added to the tree, it joins *Q*. At each stage, the adjacency list of the vertex at the head of *Q* is scanned for a neighbour to add to the tree. If every neighbour is already in the tree, this vertex is removed from *Q*. The algorithm terminates when *Q* is once more empty.

Algorithm Of Breadth-First Search

```
procedure bfs (v)
q: = make_queue()
enqueue (q, v)
mark v as visited
while q is not empty
        v = dequeue (q)
        process v
        for all unvisited vertices v' adjacent to v
                mark v' as visited
                enqueue (q, v')
```

(Thomas H Cormen et al, 2001).

4.2 Depth-first Search

Formally, DFS is an uninformed search that progresses by expanding the first child node of the search tree that appears and thus going deeper and deeper until a goal node is found, or until it hits a node that has no children. Then the search backtracks, returning to the most recent node it had not finished exploring. In a non-recursive implementation, all freshly expanded nodes are added to a last- in-first- out (LIFO) stack for expansion.

Algorithm Of Depth-First Search

```
dfs (graph G)
{
list L = empty
tree T = empty
choose a starting vertex x
search (x)
while (L is not empty)
remove edge (v, w)from end of L
```

```
if w not yet visited
{
add (v, w) to T
search (w)
        }
}
search (vertex)
{
visit v
for each edge (v, w0
add edge (v, w) to end of L
}   (Thomas H Cormen el al, 2001)
```

4.3 Depth-limited Search

Like the normal depth-first search, depth-limited search is an uninformed search. It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search. Even if the search cold still expand a vertex beyond that depth, it will not do so and thereby it will not follow infinitely deep paths or get stuck in cycles.

Algorithm Of Depth-Limited search

```
DLS (node, goal, depth)
{
        if (node = = goal)
        return node;
else
{
stack ;= expand (node)
while (stack is not empty)
{
node' := pop (stack);
if (node' . depth () < depth);
DLS(node', goal, depth);
Else
;// no operation
                }
        }
```

4.4 A* Search

A* (Pronounced 'A star') is a tree search algorithm that finds a path from a given initial node to a given goal node. It employs a heuristic estimate that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. The A* algorithm is therefore an example of a best-first search (Hart P.E. et, 1968).

Algorithm Of A* Search

```
function A* (start, goal)
        var closed := the empty set
        var q := make_queue 9path (star))
        while q is not empty
                var p:= remove_ first (q)
                var x:= the last node of p
                if x in closed
                        continue
        f x= goal
                return p
        add x to closed
        foreach y in successors (p)
        if the last node of y not in closed
                enqueue (q,y)
```

5 Results

The results of complexities measurement for each of the above tree search algorithms implemented using C, C++, Pascal, and Visual Basic programming languages are presented as shown in Tables 2 – 5 bellow.

Table 2: Results For Breadth First Tree Search Algorithm

LANGUAGES	PROGRAM VOL (V)	PROGRAM DIFFIC (D)	PROGRAM EFFORT (E)	CYCLOMATIC NUMBER
C	733	20	14660	5
C++	723	18	13014	5
PASCAL	558	17	9486	3
Visual BASIC	1045	22	22990	6

Table 3: Results For Depth-First Tree Search Algorithm

LANGUAGES	PROGRAM VOL (V)	PROGRAM DIFFIC (D)	PROGRAM EFFORT (E)	CYCLOMATIC NUMBER
C	459	20	9180	5
C++	481	21	10101	5
PASCAL	454	11	4994	5
Visual BASIC	883	15	13245	6

Table 4: Results For Depth-Limited Search ALGORITHM

LANGUAGES	PROGRAM VOL (V)	PROGRAM DIFFIC (D)	PROGRAM EFFORT (E)	CYCLOMATIC NUMBER
C	595	21	12495	5
C++	544	19	10569	5
PASCAL	626	14	8764	5
Visual BASIC	1297	22	28534	7

Table 5: A-Star Search Algorithm Complexity Measures

LANGUAGES	PROGRAM VOL (V)	PROGRAM DIFFIC (D)	PROGRAM EFFORT (E)	CYCLOMATIC NUMBER
C	459	20	9180	5
C++	481	21	10101	5
PASCAL	454	11	4994	5
Visual BASIC	883	15	13245	6

6 Conclusion

It was observed from the results of implementation (Tables 2 – 5) that Pascal programming language perform best for implementing breadth-first search, depth-first search, and depth-limited search algorithms, while C programming language is the best implementation language for A* search algorithm. We therefore conclude from these results that Pascal programming language is the best language for implementing breadth-first search, depth-first search, and depth-limited search algorithms but C language is the best implementation language for A* search algorithm.

Furthermore, it is apparently concluded that the choice of programming languages affects the complexities of programs of tree search algorithms.

References

Akanmu, T.A.(2009): An explanatory study of software complexity measure of Breadth-first search Algorithm. Journal of Science& Technology vol 1

Alfred, V.A., John, E, and Jeffrey, P.U(1974): The Design And Analysis Of Computer Algorithms. New York. Addison-Wesley Publishing Company.

Floyd, R.W. (1962): "Algorithm 97: Shortest Path" Communication of the ACM5 (6) 345. DOI: 10. 1145/367766.368168.
Geer, Daniel et al (2003): Cyber security: The cost of monopoly(PDF)2004

Halstead, M (1977): "Elements of software science, operating, and Programming systems" series volume 7. New York, NY: Elsevier, 1977.

Hart, P.E. Nilsson, N.J, and Raphael, B.(1968): Correction to: "A Formal For the Heuristics Determination Of Minimum Cost Paths", SIGART Newsletter 37: pp. 28-29

McCabe, T.J.(1994): Software complexity. Crosstalk, vol 7, no 12.
Okeyinka, E.A. (2000): Design of a scan Machine for complexity Measure of computer program. Journal of Science & Technology vol 1 No 1 pp70-77

Olabiisi, S.O.(2005): ''Universal Machine For Complexity Measurement Of Computer Programs'' PhD Thesis, Department Of Pure And Applied Mathematics, LAUTECH, Ogbomoso.

Oman, P and Hagemeister, J.: Construction And Testing of Polynomials Predicting Software maintainability Journal Of System And Software 24, March 1994: 251-266

Robert E. Park(2006): Software Size Measurement: A framework For Counting Source Statements. Technical Report CMU/SEI-92-TR-20

Russell, S.J., Norvig P (2003): Artificial Intelligence: A modern approach, pp. 97-104

Shola, P.B.(2008): Data structure And Algorithm Using C++. Reflect Publishing house, Ibadan.

Thomas, H.C. (2000): Introduction to algorithms. McGraw-Hill companies, New York.

SESSION

BATTLESPACE REPRESENTATION FOR AIR, SPACE, AND CYBER

Chair(s)

Dr. Vince Schmidt

Exploring the Fourth Dimension: a Computer Scientist's Experience in Temporal Display Development

Patrick Dudenhofer
Air Force Research Laboratory, 711 HPW/RHCI, Wright-Patterson AFB, Dayton, OH, USA

Abstract - *A new interface paradigm is required for effective supervisory control of multiple unmanned vehicles by a single operator. A temporal-based display is potentially useful for these time-critical cognitively demanding task environments. However, designs of candidate temporal interfaces for unmanned vehicles have been limited and related publications have not described the approaches used by computer scientists in the development of these interfaces. Recently, a temporal display was designed and implemented that will soon undergo extensive evaluation. This paper documents the design decisions made in the course of developing this interface in regards to the underlying temporal data structures and how they are organized and updated.*[1]

Keywords: temporal display, timeline, unmanned vehicle

1 Introduction

When directing remote platforms such as ground robots or unmanned air vehicles, the supervisory control environment is usually designed around a geospatial display. Tasks are based on the location where they are to be accomplished and generally the operator's primary job is to maneuver the platforms from one location to another. This style of command and control works well until time constraints are introduced into the task. A geospatial display cannot directly convey the time a platform will reach a specific location or how much time a planned maneuver will take to complete. When an operator controls semi-autonomous vehicles, it is not enough to simply know where the vehicles may be located in space – the first three dimensions – one must also be aware of how the vehicles will operate in time – the fourth dimension.

An operator could estimate temporal information by using the platform's speed and distance to a specified point, but that mental calculation can be time-consuming and prone to errors. However, if a supervisory control environment made use of a temporal display, such as a timeline, as well as the geospatial display, maneuvers and decisions with temporal constraints may be much easier for the operator to evaluate and execute. Such an interface would be especially useful for applications in which a single operator is supervising multiple unmanned vehicles that are collaborating on tasks. For example, if the initiation of one vehicle's task depends on another vehicle completing a task, a temporal visualization of task progression for all vehicles on a single interface should be useful for the operator to recognize conflicts.

Presentation of information on a timeline has been considered for numerous military applications. For example, a temporal display was found to facilitate scheduling performance compared to a typical geospatial display in a simulated weapon-target scheduling task for a single-ship naval warfare scenario [1]. A coordinated suite of timeline visualizations improved the speed and quality of dynamic replanning solutions in an Air Force airlift operations center and current plans are to integrate this concept in the next system upgrade [2].

In contrast, a review of the literature suggests that very few laboratories have examined temporal displays for unmanned vehicle control. Moreover, the evaluations of this concept are limited [3,4] and specifics of interest to a computer scientist implementing such a display are not documented. This prompted a research team at the U.S. Air Force Research Laboratory (AFRL) to systematically note design decisions while developing a new dynamic temporal interface for multi-vehicle supervisory control. As a computer scientist on this team, my contribution is to report on some of the design issues I encountered while developing the data structure to support this effort. Unfortunately, it is not possible to compare our team's approach with previous efforts, as their corresponding reports do not contain pertinent details. Therefore, the information provided in this paper is, to our knowledge, the first instance such design issues are discussed. Beyond informing other computer scientists, this paper describes an approach that has proven successful to date, thus providing a means by which future alternate approaches can be compared.

The recent efforts at AFRL involve a series of design and evaluation efforts with the goal of developing a temporal interface that integrates mission and vehicle information on a dynamic temporal display and also provides control functionality by which the operator can interact with many

[1] Document approved for public release 88ABW-2011-2149 (11 April 2011)

automated functions via interaction with overlaid symbology elements. Initial efforts involved evaluation of symbology options using interface formats that were static in nature [5]. Such evaluations were limited as having the symbology move dynamically, as time progresses, may influence the operator's ability to retrieve information. Also, these evaluations utilized part-task simulations in which the experimental participants' only task was to retrieve information from the temporal interface to answer questions [6]. For many control applications involving multiple vehicles, operators will be required to switch attention across multiple tasks, some employing the temporal format and some with other interfaces in the control station (e.g., communications and imagery inspection). Thus, a temporal interface that is dynamic and used in conjunction with a high-fidelity simulation of the entire task environment is required so that the experimental subject's workload represents the future vision of multi-vehicle supervisory control applications.

The recently implemented dynamic temporal interface at AFRL will soon undergo extensive human factors and usability evaluations. The first experiment will examine whether or not the initial temporal display design increases operator situation awareness, reduces workload, and improves task performance. Experimental trials will be completed with and without the prototype temporal interface, using test scenarios that require completion of multiple tasks envisioned for the application environment. Subjective data will also be collected to determine changes recommended in the temporal interface. Subsequent experiments are then planned, with each one involving a newer version of the temporal interface, incorporating additional information display and control functionality.

The development of AFRL's initial dynamic temporal interface first required several design considerations and decisions with respect to the required data structures in order for the product to support the changing realities inherent in a multiple vehicle control system. As a computer scientist member of this research team, I will introduce in this paper some of the design decisions associated with the temporal data structures needed for various temporal displays. A number of these decisions reflect thinking through the issues surrounding the manipulation of temporal information. Most, however, reflect decisions arrived at via trial and error during the temporal interface design process. AFRL's prototype (version 1) temporal interface for unmanned vehicle control has recently undergone preliminary checkout and the results indicate that the data structure successfully supports its use for application in the simulated vehicle control environment (i.e., performs well with no noticeable delays in interface responsiveness). Therefore, it is timely to document the decisions made in the course of developing this interface pertaining to the design of the underlying temporal data structures and how they are organized and updated.

2 Viewing the Fourth Dimension

There are several ways that temporal information can be displayed to an operator of a multi-vehicle control system. The two most common methods are the ordered-list view and the timeline view. The ordered-list view is a list of temporal events associated with a time and possibly a vehicle and duration [7] (Figure 1). This type of display is usually sorted chronologically but could also be sorted by vehicle or task type. The other basic type of temporal visualization is the graphical timeline [4,8] (Figure 2). The system's temporal data may be represented by a single line augmented with overlaid symbology for each event associated with it along the line. The distance between each symbol on the line is proportional to the time between each temporal event associated with those symbols. For multi-vehicle control, with

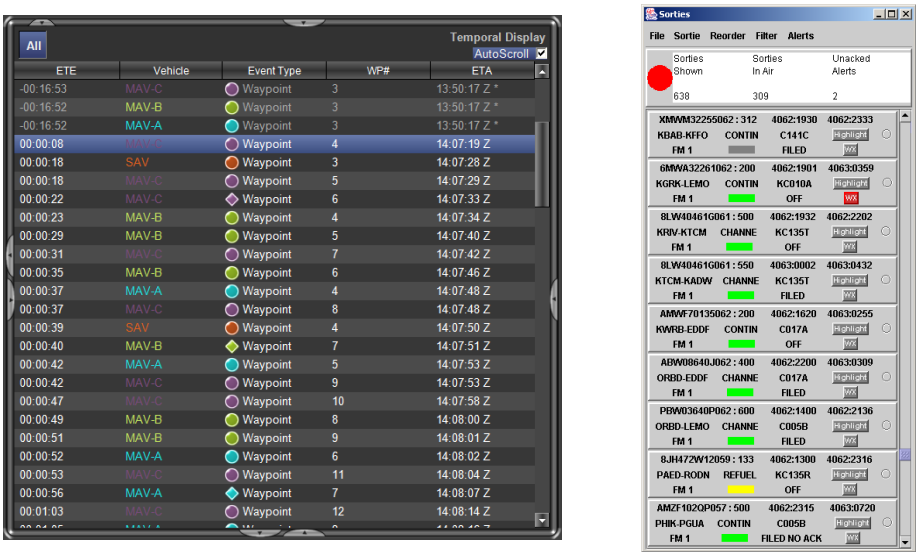


Figure 1. Examples of a ordered-list view for temporal information. *(Left: Prototype interface; Right: GAMAT interface [7])*

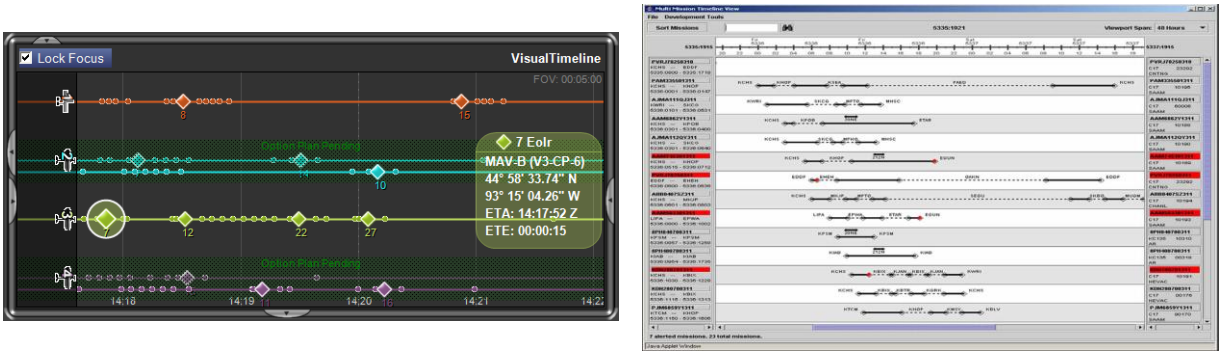


Figure 2. Examples of a timeline view for temporal information. *(Left: Prototype interface; Right: WIDE interface [8])*

multiple lines, each line can represent a vehicle controlled by the system. This view more clearly associates the temporal data with a specific vehicle and allows for easier comparisons between each vehicle's upcoming schedules. It is important to consider how the operator will view and manipulate the temporal data when supervising multiple semi-autonomous vehicles as this should affect design decisions for how the underlying temporal data structures for the system should be organized and updated.

3 Future versus Reality

When designing a temporal interface for semi-autonomous vehicles, one of the first problems to consider is the practical difference between simulated time and actual time. It is impossible to model time and future events for a system perfectly. For instance, an algorithm may calculate that a vehicle will arrive at a certain point at 5:45PM but in actuality the vehicle could arrive at 5:47PM due to a strong headwind that is not being modeled by the calculations. It is impossible to track and display the real time of arrival if the mathematical model doesn't know about the strong headwind slowing down the vehicle. The lack of complete and thorough information prevents a system from perfectly projecting the future actions and progress of the remote platform. A decision must be made on how to keep the system accurate as time progresses.

A similar issue arises when using a temporal interface as a tool for time manipulation. If the temporal display is to be used to not only view the current state of the platforms but also to direct and manipulate the future state, how will the operator effectively distinguish between the actual data and the newly modified but still pending data? One option is to obscure the actual temporal data and only display the newly modified temporal information until the new temporal information is finalized - at the possible expense of hindering the operator's temporal awareness of current events. Alternatively, a second identical display could be used to manipulate the temporal data where the actual real data is always visible in the primary temporal display. For the software engineer a similar decision must be made in both cases – when displaying originally-planned data while showing up-to-date temporal information and when viewing current temporal data while manipulating future possible temporal information – which temporal information is more important to the operator for the primary view and how can one differentiate between the various types of temporal data?

4 Capturing the Fourth Dimension

One goal of a data structure driving temporal displays is to allow any potential display to use the same underlying data without conflict. In essence, all the data should be stored and computed independently and in the background while any display or algorithm simply uses the data without having to perform any calculations itself. Any temporal information should be available immediately upon request without need for further computation. Separation between the interface and the data is an important concept in software engineering and is certainly useful in this domain.

Events that occur can be thought of in two different ways: 1) instantaneous events that occur at a point in time with a negligible duration or 2) events with a significant duration that begin at a particular time and end at some point in the future. If an autonomous vehicle is traveling to a waypoint, the vehicle's arrival at that location can be thought of as an instantaneous event. The vehicle is not arriving at the location until it actually gets there and once it gets there it is no longing arriving – it has already arrived at the point and that event is in the past! Other vehicle events have an associated duration. If a vehicle is tasked to search an area, it begins the task, continues its searching for a finite amount of time and then stops. What is the best way to order these two types of events in a data structure? If every event was instantaneous it would be an easy decision – order the events by time. Where does one fit durational events in a data structure? Does a durational event come before an instantaneous one even if the event with duration ends after the event that occurred in an instant? In the case of an event with duration, instead of one item, perhaps the beginning and end of that event could be represented as two separate instantaneous events with a link to each other. It is probably

better to represent events with the duration as a single item in a way similar to the instantaneous events but with the duration information associated with them as well.

The same considerations will have an impact on the display of both kinds of events as well. Instantaneous events can be displayed by a single point on a timeline. Events with duration must be displayed by at least two points on a timeline connected with some distinguishing feature. In most cases it seems best to draw events on screen from oldest to newest so that the most recent event is drawn last and, in effect, layered on top of the earlier events. In this way, if an instantaneous event occurs before an event with duration ends, the later instantaneous event is not obscured by the older event with duration (Figure 3).

Another design decision is how to organize the temporal items within a system supervising multiple vehicles. Should events across the entire system be stored together in one large data structure or should each entity in the system be responsible to store and track its own temporal information? Occasionally the operator will want access to conflated timeline information, and at other times he will only want information specific to a single vehicle under his supervision. It seems to be easier to manage and retrieve temporal information for each separate entity independently rather than search a large data structure to gather temporal information pertinent to a single vehicle. This approach keeps temporal information local to the individual vehicles and prevents unnecessary computationally-expensive searching and sorting when the data is requested by the user.

For storing this information, a list-based data structure generally is one of the best ways to store temporal data. Because of the dynamic nature of some temporal events it may be difficult to keep the list in temporal order as some of the events need to be consistently updated to reflect a dynamic reality. When data is requested from the data structure, it can be sorted on demand at that time rather than trying to place the changing events in chronological order. Attempting to keep everything continuously sorted created a noticeable computational strain on our multi-vehicle control system without providing any significant benefits.

5 Staying Current

Determining how to initialize and update the temporal data representing the system is another design decision to consider. Initially, the data structures will have to be populated from scratch using the data already present in the system. For instance, a vehicle may have a movement plan with several waypoints associated with it. The initial data doesn't tell us what the vehicle's time of arrival will be at the first waypoint, not to mention the arrival times for the following waypoints in the vehicle's plan. However, the vehicle's time en route to its first waypoint can be calculated

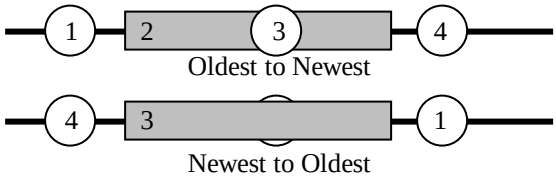


Figure 3. The effects of drawing order.

by computing the distance between the vehicle's current location and the first waypoint then multiplying the vehicle's current velocity by that distance. With the time en route already determined, it's a simple calculation to find the vehicle's estimated time of arrival at that initial waypoint. Estimated times at subsequent waypoints are much easier to determine - just find the distance between the two waypoints and multiply by the vehicle's commanded speed for that leg of the journey. As you compute the arrival times for waypoints that are beyond the waypoint that the vehicle is currently moving toward, simply add the time en route for that leg to the arrival time of the waypoint that begins that segment. In this way, the arrival time for each waypoint in the vehicle's plan can be quickly calculated based on the preceding waypoint's arrival time and then stored in the temporal data structure. Of course, all temporal calculations provide only estimates due to the effects of unknown variables such as wind, traction, etc.

5.1 Remaining Up-to-Date

Once the initial data structure is populated, the question turns to how the temporal data should be kept current. Some temporal events are not dependant on the vehicle's performance. They will occur unconditionally at a specific time and nothing will change the temporal information associated with that event. Once this static type of temporal event is added to the temporal data structure, nothing further needs to be done to keep it up-to-date with respect to reality.

In most cases, an event's temporal information is directly dependent on a vehicle's movement and actions. The distance and expected speed between two waypoints in the vehicle's plan will not change; therefore, the time en route between those points will not change either. The calculated time en route between two waypoints only needs to be computed once. However, the distance between the vehicle and the waypoint it is travelling toward is changing constantly (and ideally decreasing) because the vehicle is moving. If the temporal calculations used initially perfectly described reality, no updating of the temporal data would be necessary, but perfect simulation of the real world is impossible – a ground vehicle could lose traction or an air vehicle could be caught in an unexpected headwind or tailwind. Since we have to expect our initial calculations to provide merely an estimate that will change as time progresses, the temporal data must be periodically updated to reflect reality. In the case of a vehicle following a plan of waypoints, only the distance between the

vehicle and the upcoming waypoint has changed necessitating an update of the estimated time of arrival at that waypoint. Since the first waypoint's estimated time of arrival has changed, all subsequent waypoints' estimated times of arrival must change as well to reflect the new reality even though the estimated time en route between those waypoints has not changed (Figure 4).

5.2 Moving Forward

Another consideration concerns knowing whether or not a vehicle has actually accomplished the event modeled in the temporal data structure. In keeping with our waypoint following example, how do we know if the vehicle actually arrived at the waypoint? Our calculations for determining the estimated time of arrival for a waypoint depend only on two points in space and a current speed. Although the waypoint example is simplistic, in reality a vehicle could fail to realize that it passed over its intended destination and not continue on to the next waypoint in its plan. The vehicle hasn't yet (knowingly) met its goal in arriving at the waypoint; therefore, the temporal data structure should reflect that reality by continuing to update the estimated time of arrival by the previously described method. As the vehicle moves past its intended destination, the estimated time of arrival at that point will increase causing, in turn, all future waypoints' times of arrival to increase by the same value as well. Since the temporal data itself cannot determine if the vehicle arrived at a particular waypoint, the system will have to alert the temporal data structure when it occurs through some kind of event mechanism. Once the temporal data structure receives the indication that the vehicle has arrived at the waypoint, the temporal information can be updated to begin estimating the time en route from the vehicle to the next waypoint in the plan. Most temporal events will require some kind of notification from an outside source to inform the temporal data structure if and when the event was achieved.

6 Saving the Fourth Dimension

When supervising multiple semi-autonomous vehicles, it may be useful to not only see the upcoming events for each vehicle but also to examine past events. With the operator's attention pulled in many directions, it is very possible to miss some events. Providing a method to review previous events may allow an operator to gain greater understanding into the current situation. Fortunately, if the temporal data is stored in a list structure, keeping and retrieving past event information is relatively trivial. Once a temporal event has been completed via vehicle notification or the passage of time, the event is simply left in the list data structure. The past event will never need updating nor should it. Any user can request a list of events associated with a particular vehicle and specify whether or not the data should contain past events.

7 Performance Challenges

Since it is impossible to model reality perfectly, every piece of temporal information captured for the system control interface must be periodically updated to reflect the changing reality. If the data is updated infrequently, the operator will not have an accurate understanding of the actual status of the vehicles being supervised. However, updating too often can place a strain on the performance of entire system control interface because the temporal data structure is regularly polling for new data, listening for vehicle notifications and recalculating times for most of the temporal events being stored. This problem can be exacerbated when multiple interfaces within the system are requesting temporal data on a continual basis. The rapid recalculation of so much dynamic data can make other parts of the system sluggish or unresponsive – a situation that is not acceptable for a system representing multiple semi-autonomous vehicles in real-time. In our experience, for a system involving approximately four vehicles, an update or recalculation rate of one hertz seems to

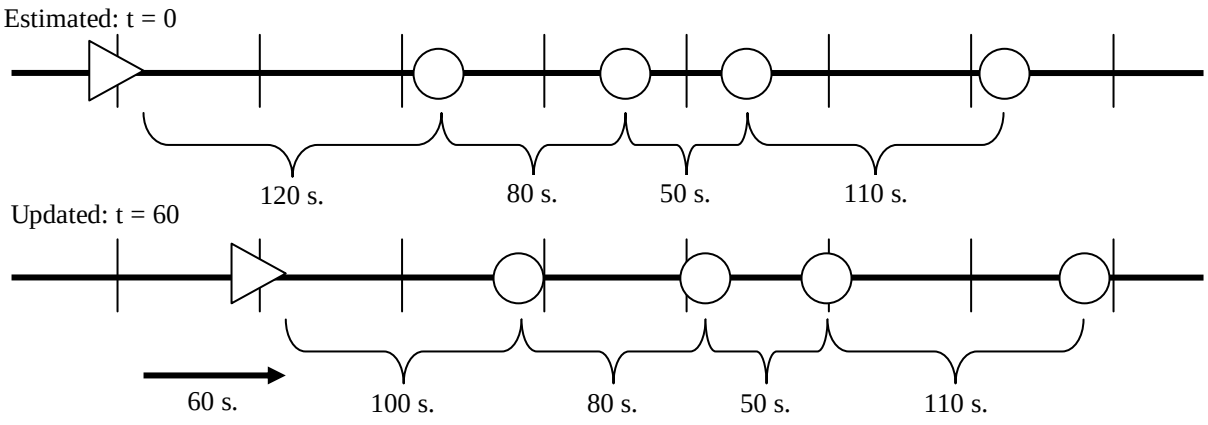


Figure 4. The vehicle has been moving for 60 seconds at a commanded airspeed but only gained 20 seconds toward its next waypoint due to a headwind. The time en route between the following waypoints is fixed even though their estimated times of arrival have been delayed.

be a reasonable compromise between stale information and an unresponsive system. The temporal information does not vary enough in one second's time to cause large deviations in the data. Thus, the representation of the temporal events appears continuous on the display to the operator. On the other hand, vehicle acceleration, if not modeled in the calculations, will cause some "time-shifting" in the temporal data – the greater the magnitude of the acceleration the greater the temporal data changes during each update. Generally, changes in estimated times will occur slowly from the operator's perspective, rarely shifting events more than a few seconds per update. For our application, one hertz allows ample time for the rest of the system to operate without undue stress from temporal data updates and calculations. Future evaluations will solicit operator comments on the adequacy of the temporal information update rate.

8 Accessing the Fourth Dimension

Once the data structures are in place to store and update the temporal data associated with each vehicle, methods are required to allow the data to be retrieved in a useful manner. Often only a portion of the temporal data is needed or desired so the ability to set bounds on what data is to be returned to the user would be useful to provide. Some visualizations may not want any historic data to be included in the request. The ability to specify which source or sources of the temporal data to retrieve is also quite valuable for the operator. A few temporal displays may require all the temporal data for every vehicle in one large list while others may want data from a single source or vehicle. Data structures should be implemented in such a way as to allow users to request data pertaining only to specific vehicles and to temporal events within a given window of time. Given that the nature of a list data structure (coupled with dynamic updating of the temporal information) does not guarantee sorted order, all data is generally best sorted chronologically when it is requested. All temporal information should also be returned to the user in a thread-safe manner to prevent unexpected changes and conflict in the data. Many problems can pop up if one tries to use and display temporal data while that same data is simultaneously being updated elsewhere.

9 Back to the Future

There are many other aspects surrounding the use of time-specific information for supervising multiple semi-autonomous vehicles that ought to be explored. One issue that was touched on in this paper was how to design a temporal display that clearly distinguishes between the current state of the system and the proposed or requested changes to the system. In addition, there will be times when the operator will need to compare the current (and future) state of the vehicles with one or more proposed alternates. Determining how best to present multiple temporal options to an operator remains a research question yet to be explored. Another issue to consider is the amount of screen real estate to dedicate to a temporal display and the effect of cluttering as the visual display presents more and more symbology. In the future there will probably be intelligent agents (background computer processes) that will be constantly reviewing and analyzing the temporal data to provide insights and suggestions to the operator regarding the progress of the vehicles' missions under their supervision. How will the design decisions made now support or hinder operators – human or computer – in the future?

10 Conclusion

Designing a temporal interface for monitoring multiple autonomous vehicles requires some attention and consideration both on the nature of time and on how an operator will need to interact with temporal information. The information provided in this paper contributes to the design of temporal interfaces by documenting design issues and approaches used in regards to the underlying data structures. Choices in how temporal information is stored, updated, and manipulated will have far reaching consequences as visualizations, algorithms, and operators make use of the available temporal data. It is only through systematic human factors evaluations, however, that the usability of the temporal interface can be confirmed, informing the adequacy of the approach for designing the temporal data structures as documented in this paper. When implemented correctly, the operator will have an effective way to supervise and control multiple autonomous vehicles as they explore – and exploit – the fourth dimension.

11 Acknowledgements

Special thanks to Ms. Gloria Calhoun and Dr. Terry Stanard for providing editorial comments.

12 References

[1] Rousseau, R., Tremblay, S., Lafond, D., Vachon, F., and Breton, R. (2007). Assessing temporal support for dynamic decision making in C2. *Proceedings of the Human Factors and Ergonomics Society,* 1259-1262.

[2] Scott, R., Roth, E., Truxler, R. and Wampler, J. (2009). Techniques for effective collaborative automation in dynamic air mission planning. *Proceedings of the Human Factors and Ergonomics Society*, 202-206.

[3] Cummings, M.L., Brzezinski, A.S., and Lee, J.D. (2007). The impact of intelligent aiding for multiple unmanned aerial vehicle schedule management. *IEEE Intelligent systems: Special Issue on Interacting with Autonomy*, *22*(2), 52-59.

[4] Cummings, M.L. and Mitchell, P.J. (2005). Managing multiple UAVs through a timeline display. *Proceedings of the Information Technology @Airspace*, AIAA-2005-7060.

[5] Spriggs, S., Warfield, L., Calhoun, G., and Ruff, H. (2010). Orientation of a temporal interface for multi-unmanned aerial vehicle supervisory control. *International Conference on Human-Computer Interaction in Aerospace (HCI-Aero)*, Cape Canaveral, FL.

[6] Stanard, T., Dudenhofer, P., Spriggs, S., Calhoun, G., Warfield, L., and Ruff, H. (2011). Portraying the passage of time in a timeline interface for supervisory control. *Proceedings of the 16th International Symposium on Aviation Psychology*.

[7] Kuper, S.R., Scott, R., and Eggleston, R.G. (2003). Using work-center support system technology to enhance command and control. *Proceedings of the 8th International Command and Control Research and Technology Symposium*, Washington, DC.

[8] Roth, E., Stilson, M., Scott, R., Whitaker, R., Kazmierczak, T., Thomas-Meyers, G., and Wampler, J. (2006). Work-centered design and evaluation of a C2 visualization aid. *Proceedings of the Human Factors and Ergonomics Society 50th Annual Meeting*, 255-259.

A Semi-automated Display for Geotagged Text

Vincent A. Schmidt
Air Force Research Laboratory
Dayton, Ohio USA

Jane M. Binner
Sheffield Management School
University of Sheffield, UK

Abstract

The changing dynamic of crisis management suggests that we should be leveraging social media and accessible geotagged text data to assist with making emergency evacuations more effective and increasing the efficiency of emergency first responders. This paper presents a preliminary visualization tool for automatically clustering geotagged text data, and visualizing such data contextually, graphically, and geographically. Such a tool could be used to allow emergency management personnel to quickly assess the scope and location of a current crisis, and to quickly summarize the state of affairs. Discussion herein includes details about the clustering algorithm, design and implementation of the visualization, and ideas for improving the utility for use in a variety of circumstances. [1]

Keywords: visualization, geotagged text, social media

1 Introduction

The dynamics of crises are changing. In particular new communications and social networking technologies mean that unprecedented opportunities for real-time communication in evacuation at street level are emerging. Public information for warning of impending crises need no longer be transmitted in a 'top-down' fashion and rapidly we have moved to a situation where public information is just one signal in an information market place. Victims and evacuees can now communicate with each other before, during and after an emergency and gain real-time access to information. As a result, they may indulge in individual and collective strategic behavior (by generating alternative information and rumors perhaps). Such strategic decisions by evacuees can be analyzed to design safer and faster evacuations. In order to do so public information and planning needs to become interactive, dynamic and responsive. For example, it is plausible for emergency management agencies to collect information during a crisis (e.g. messages posted on social networking sites) and to use this, not only in managing evacuations, but also to intervene by posting messages, selectively targeting trusted sources and modifying electronic signage. This would modify incentive structures in evacuation through acting on real-time information.

One way to promote this capability is to design visualizations for emergency management personnel that are capable of displaying relevant data about the contents and metadata within real-time social media messages. Such systems should summarize salient semantic points in such a way that first responders could re-inject recommendations back to social media and emergency management communications systems in near real time.

The uses for such visualization reach beyond field use by first-responders, though. Similar systems could be used to monitor general community "health," acting as an additional input into crisis prediction and recognition scenarios. Social scientists and analysts could also use such systems to investigate causal activities and understand their impacts.

This paper provides the basis of a preliminary study to examine the utility of visualizing geotagged text data, similar to that available through social media, as a tool for analysts, policy makers, first responders and crisis management personnel, and social scientists. After summarizing certain background information, we describe the contents of a typical dataset and discuss how it could be prepared and demonstrate how it is used in straightforward geospatial visualizations. We conclude with comments about the types of visual-

[1] Paper cleared for public release.

ization we believe might be most appropriate, and indicate interesting areas for future work.

2 Background

A number of synoptic and small-scale studies have already noted the use of individual electronic communication including the Californian wild fires of 2007 [1] and the 2007 Sheffield floods [2]. The role of agents and computational models in crisis management has been considered by Chen and Xiao [3] who consider that real-time information can give feedback resulting in the adjustment of plans by the emergency services. Innovatively, Nakajima et al [4] have considered the use of ubiquitous devices such as GPS and mobile phones to build a multi-agent evacuation strategy for the city of Kyoto, whilst Ushahidi [5] maintains up to date reports submitted by the public and makes available latest incident reports on their website to assist victims of the Haiti earthquake. There is, of course, a large body of existing work on mathematical modeling of evacuations [6–8], but not much work has been published on the role of feedback loops and social networks in evacuation. Our project is one of the first to systematically combine emergency planning and visualization simulation of crisis behavior taking into account exchange of information through social networking, and considering the resulting aspects of strategic decision making.

Given the difficulty of conducting real-world experiments of crisis behavior it is hard to make valid inferences as to the effects of social network technologies without advanced computational modeling and systematic validation and testing. Similarly without computational modeling it is difficult to identify the ways in which responders may intervene in such networks. Furthermore, there is a need to involve policy makers and responders in the validation of these models. Governmental organizations are already interested in social networking and communication technologies for resilience. For example, Twitter has been noted in earthquake prediction (http://twitter.com/quakeprediction), is used by the Los Angeles Fire Department (http://www.govtech.com/gt/579338) and the Cabinet Office is considering how mobile technologies could be used in warning and informing the general public.

An impressive demonstration of the importance of ideas of collective behavior in connection with the use of social networking technology, is the recent 'balloon hunt' by the Defense Advanced Research Projects Agency (DARPA), a Pentagon agency in the US (see [9] for an account in the recent press). Further details of new research into developing tools to model collective behaviour from the theory of complex adaptive networks involving the inoculation of networks with information, and advance agent-based models of emergency planning allowing for emergent communication channels are available in [10]. This project will address these new challenges through a systematic computational modeling approach.

We desire to analyze the behavior of populations in a crisis and evacuation, focusing on the effect of receiving, spreading and acting on information on the behavior of agents/evacuees. Based on these data visualizations an understanding will be developed of how the behavior of agents/evacuees can be modified and controlled through the use of real time intervention in social networking and communications technologies. The main outputs of the project will be the speedier identification of intervention strategies for more effective crisis management, thereby making evacuations safer and faster. Ultimately, we will make recommendations to stakeholders as to the efficiency of different communication channels and control strategies arising from our simulations. This will provide a sound basis for policy makers and responders to strategize about intervention in communication and social networking technologies.

3 Dataset Preparation

The dataset used in this preliminary concept is drawn from the 2011 VAST Challenge, related to the 2011 IEEE Conference on Visual Analytics Science and Technology (IEEE VAST). The mini-challenge 1 dataset (Geospatial and Microblogging — Characterization of an Epidemic Spread) is interesting to us because the data consists of a million uniquely identified records containing originator id, timestamp, geospatial information, and message text. This information is typical of social networking data obtainable from SMS text messages, Twitter, and other commonly used sources.

The VAST (mini-challenge 1) problem is to use the text records, coupled with information about a fictional city (its population, industry, hospitals, weather, and transportation modes), to determine the source of an epidemic disease and how it is spread throughout the community.

Instead of using the data to solve the VAST challenge, our immediate interest is examining how this type of information can be effectively displayed to a specific end-user or analyst. To that end, 10000 records were randomly selected from the original dataset, provided as a comma-separated value (.CSV) file, for evaluation. Figure 1 shows the header line of our resulting data file, and includes a small representative selection of records. ID is a unique identifier for a specific user, Created_at is the record timestamp, Location is a Lat/Lon pair indicating the geographic source of the message, and the body of the message is last.

Casual examination of this small set of examples promotes the correct conclusion that the messages included in the dataset do not all relate directly to the question posed by the challenge problem. Of course, this is typical of such data. It is also easy to see that the messages may contain numerous typographical and punctuation irregularities, acronyms, and various shorthand symbology. Note that the record content is not limited to the English language (and for VAST, there are many foreign language messages in the corpus).

There are several geotagged datasets available for research use, and they are generally fairly large. In reality, this type of data is practically infinite in size, since it can often be streamed in real time directly from its source. Trying to graph or display all of this data simultaneously is obviously burdensome to the system and overwhelming to the user. Therefore, the display must be designed to support the user's work needs.

Visualizing a large quantity of messages effectively is best accomplished by dividing the dataset into smaller and manageable message groups. We accomplish this by sorting the records by timestamp, then grouping the results according to messages that are temporally "close" to one-another. We believe that "conversations" can be found by looking for messages that happen in close temporal succession, and these conversations are a reasonable way to begin to select message subsets. (Admittedly, there are many other ways to group this data.) Groups of messages in a conversation can be further subdivided geographically, yielding a finer level of detail for the analyst.

The method used to divide, then subdivide, the collection of messages is based heavily on the automated clustering algorithm developed by Schmidt [11] for preprocessing neural network datasets. It is important to note that the automated algorithm does not require any user intervention to generate reasonable clusters, and is not based on fixed-length or fixed-time strategies for dividing the data. This is best shown by example.

Figure 2 depicts the basic technique using 50 randomly generated uniform data points in (0,1000). Figure 2(A) shows the original data. The algorithm sorts the data (2(B)), then finds the difference between consecutive data points. The differences for this dataset are shown in 2(C). The algorithm slices the data at the positions where the difference is greater than twice the standard deviation of the differenced data. (The standard deviation is shown as a horizontal red line in this Figure). The observant reader will notice the "steps" in the sorted data of Figure 2(B) are located directly above the most prominent differences indicated in Figure 2(C). These are the same places the data cluster boundaries are created.

Clustering our geospatial data is a multistep process. The first step is to find clusters based on the distance from some arbitrary point. Our system currently selects the center of the map, and messages are sorted and clustered based on their distance from the center. Then, each cluster is examined and, possibly, subdivided again, depending on the (angular) location of the messages in relation to the center of the given cluster. The same algorithm is reused for this subdivision, but the "distance" measure is the relative angles separating message locations.

Although this approach yields generally useful geospatially clustered data, it is unlikely that using geospatial indicators is the best approach to clustering cyber data such as social network messages, tweets, SMS text messages, and similar types of information. To address this shortcoming, we are working on adding semantic clustering algorithms as an addition to (or alternative for) the existing technique.

```
ID,Created_at,Location,text

3,5/18/2011 13:26,42.22717 93.33772,

      this convention filled with technology could be better don't plan on leaving
      anytime soon

57,5/12/2011 21:08,42.23363 93.34164,

      There's no point of trying if no one else i s...

73,5/16/2011 5:28,42.28818 93.33605,

      sick of hearing bout the Oil Spill now like

127,5/10/2011 16:53,42.28051 93.34164,

      I hope #inception is as good as everyone s ays  ten bux is ten bux...

238,5/9/2011 14:13,42.21771 93.33845,

      ...I agree  the most demanding task in our  time is come to terms with space;
      so let time be our personal unit of confusion. ..

464,5/4/2011 20:04,42.28523 93.44908,

      Isnt it weird that after all the hoopla abo ut the TSA there was another
      'near disaster' averted? Seems like it happens too  much.

592,5/17/2011 3:27,42.28818 93.28493,

      Being sick is exhausting or I'm exhausted b /c I'm sick. Either way
      a bed would be nice right now.
```

Figure 1: Selected Dataset Records

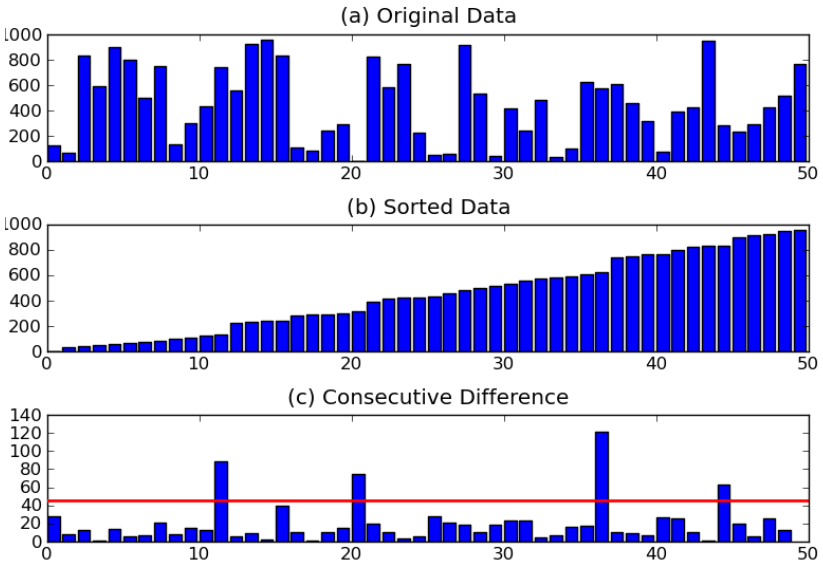


Figure 2: Automated Clustering Example

4 Visualizing the Data

Our primary purpose for visualizing geotagged textual data is to provide first responders and other analysts a mechanism for quickly identifying and responding to trends in social media data that indicate the status of a catastrophe or crisis. When the visualized data has no real form, and looks like "noise" from a variety of sources, it is most likely that no crisis is underway, so no immediate action is required. If, however, the data tends to converge to include specific topics of interest, or many messages are suddenly from (or about) a certain geographic location, a catastrophe or crisis requiring specific action may be under way. Good visualization of such data is expected to enhance an analyst's ability to quickly offer relevant guidance to the appropriate authorities and catastrophe response personnel.

The visualization we are currently considering explicitly divides the display into three sections: message relationships, geospatial information, and message clusters. Figure 3, derived from the VAST data, is typical of this display.

The top portion of the figure represents the message relationships. It is a graph, partitioned into (automatically) geographically clustered sets of messages. Clusters are labeled alphabetically, starting with the letter "A," and messages within clusters are numbered from 0. In this figure, message nodes are linked in monotonically increasing time, such that A0 is timestamped before A1, which is timestamped before A2, etc. This arrangement of links is certainly not ideal, however. It would probably be more valuable to link the message nodes based on the semantic similarities of the contents of the message bodies. Work to revise the message graphs in this manner is in progress.

The center of the figure contains a map of the geographic area of interest. A line connects the message graph to the mean location of all messages within that cluster. The actual position each message is indicated by the corresponding message label on the map, and the mean location of the message cluster is indicated by the placement of the cluster label on the map. We can optionally draw a line from each message node on the graph to its location on the map, but we found this additional information clutters the display and makes it tedious to read.

On the bottom of the figure is a histogram representing (on the X axis) the number of message clusters (actually 406 in this case, not explicitly indicated on the display), and (on the Y axis) the number of messages in each cluster. The red mark on the histogram indicates the message cluster currently being graphed and shown on the map.

The user can change the currently displayed cluster by clicking the mouse on the histogram, or by using the cursor keys to move left or right by a single cluster. Hovering over a graph node at the top of the figure shows a tooltip containing the text of the selected message.

Other visualization are being added to the display as the semantic analysis and related development continues.

5 Conclusions & Future Work

There are many ways to display geotagged text data. However, the effectiveness of the visualizations depends heavily on the objectives of the end-users. Users with different work requirements will often need different types of interfaces, even if they use identical data sources.

The visualizations we demonstrate in this effort are entirely exploratory in nature. We perceive a variety of uses for this basic type of interface:

- *Fully automated operation* might watch one or more social media streams in real time, indicating an alarm condition if a certain subset of words becomes frequent, or if a particular location is referenced in a certain way. This would require the addition of flexible filters to the existing design. Such a mode would be valuable to first responders as an additional watchdog for catastrophic events.

- *Query design* would allow a user to type a textual query, and the message graphs would be reorganized depending on the semantic contents of the query. A more structured interface would have to be added to make this utility viable. This mode promotes a more visual geospatial search operation.

- *Exploratory visualization* occurs when the records are displayed using self-organizing algorithms. Deciding the "interestingness" of data is the grail of data mining, but a growing

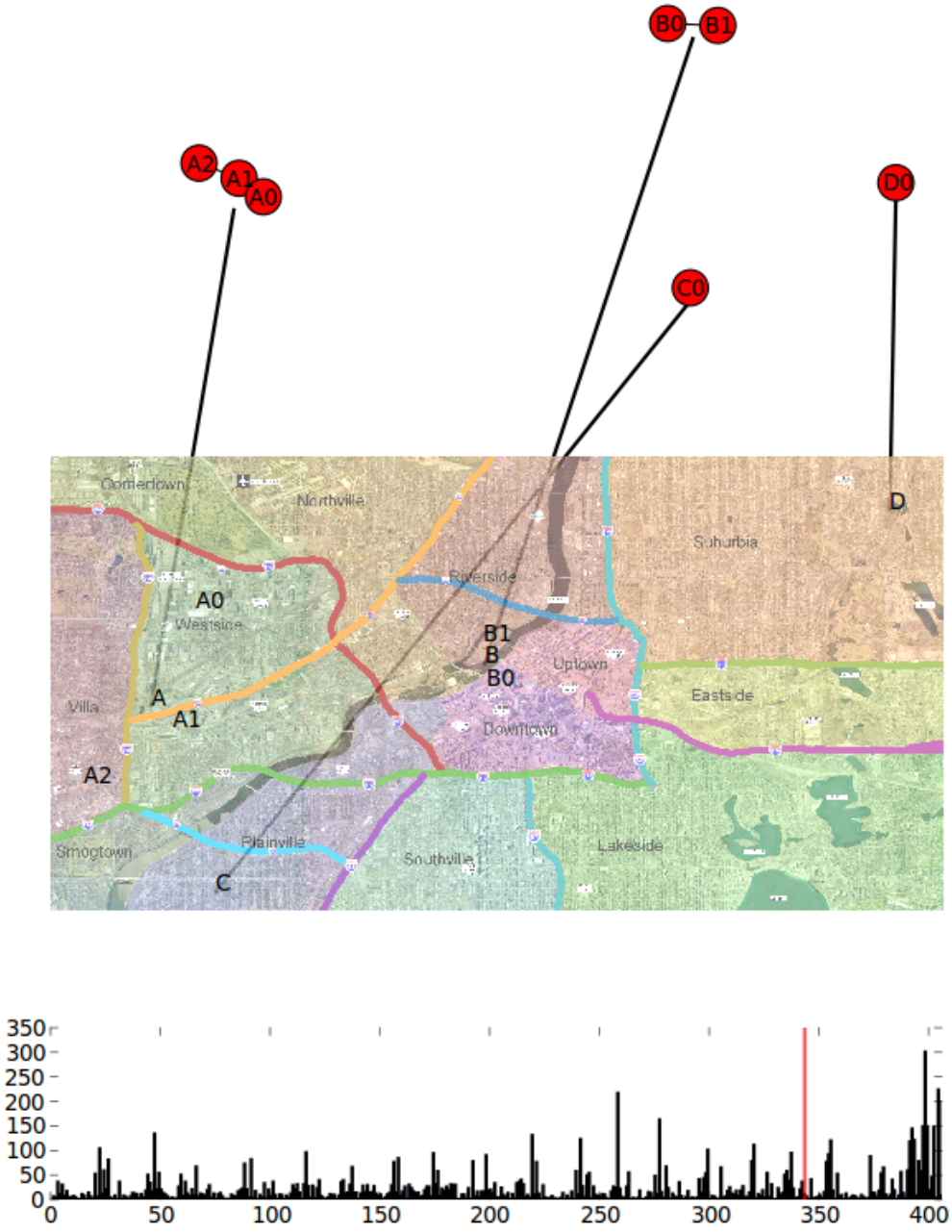


Figure 3: Visualizing Geotagged Data

collection of algorithms and visual interfaces allow analysts and scientists to leverage these systems as tools more easily, especially when the automation can be trusted to identify and display certain trends without explicit interaction. For now, the utility we demonstrate requires direct control by a user.

Our short-term plans include the incorporation of a variety of semantic algorithms, permitting the message graphs to be connected in more meaningful ways. This enhancement would also allow the graph clusters to be tagged to support semantic search operations.

Adding a search tool or a watch-list of interesting terms would enable the utility to be used to display the results of simple searches. One simple extension to this concept is to collect geotagged results from commercial search engines, then use

the visualizations we describe here to display those results.

Although our initial implementation is designed for traditional computer screens, we have access to a sizable collection of advanced 3-d technologies. It would be interesting to revise the visualizations such that these technologies could be used when available. It would also be useful to experiment with adding these visualization concepts to small displays, such as cell phones and the latest generation of tablet computers.

Such a move is in keeping with other similar developments such as the new national alert system which is set to begin in New York City to alert the public to emergencies via cell phones. This new Personal Localized Alert Network (PLAN) will enable presidential and local emergency messages as well as Amber Alerts to appear on cell phones equipped with special chips and software. The Federal Communications Commission and the Federal Emergency Management Agency confirm that the system will also warn about terrorist attacks and natural disasters.

There is clearly much work to be done. The goal is far more important than the mere display of message data on a graph or map. The ultimate objective is to create a reliable tool that allows first responders and others to leverage social media to protect the public at large. The testimony of the value of such a tool occurs when those who use the prototypes designed for their work areas are able to claim these devices and visualization are directly responsible for lives being saved.

6 Acknowledgements

The authors would like to acknowledge the generous support of EPSRC Grant reference number EP/I005765/1 for the funding provided and to the members of the EPSRC Dfuse team, particularly John Preston, University of East London UK and Maria Angela Ferrario, University of Lancaster for research support.

References

[1] J. Sutton, L. Palen, and I. Shklovski. Backchannels on the front lines: Emergent uses of social media in the 2007 southern california wildfires. In *Proceedings of the 5th International ISCRAM Conference.*

[2] V. Lanfranchi and N. Ireson. User requirements for a collective intelligence emergency response system. In *Proceedings of 23rd BCS HCI Group conference (HCI 2009).*

[3] Chen and Xiao. Real-time traffic management under emergency evacuation based on dynamic traffic assignment. In *Proceedings of the IEEE International Conference on Automation and Logistics (ICAL 2008).*

[4] Y. Nakajima, S. Yamane, H. Hattori, and T. Ishida. Evacuation guide system based on massively multi-agent systems.

[5] Ushahidi. http://haiti.ushahidi.com/main, 2009.

[6] D. Helbing, I. Farkas, and T. Vicsek. Simulating dynamical features of escape panic. *Nature*, 2000.

[7] C. Burstedde, K. Klauck, A. Schadschneider, and J. Zittartz. Simulation of pedestrian dynamics using a two-dimensional cellular automaton. *Physica A 295*, 2001.

[8] A. Ferscha and K. Zia. Lifebelt: Crowd evacuation based on vibro-tactile guidance. *IEEE Persvasive Computing Issue*, 2010.

[9] M. Hesse. Mit team wins social networking balloon hunt. *Washington Post*, 2009.

[10] J. Preston, L. Branicki, MA Ferrario, and M Kolokitha. Multiple attacks on transport infrastructure: an inter-disciplinary exploration of the impact of social networking technologies upon real time information sharing, response and recovery. *Journal of Homeland Security (forthcoming)*, 2011.

[11] Vincent A. Schmidt. *An Aggregate Connectionist Approach for Discovering Association Rules.* PhD thesis, Wright State University, Dayton, OH, May 2002.

SESSION

SOFTWARE MEASUREMENT - THEORY AND PRACTICE

Chair(s)

Dr. Austin Melton

WAESM: Web Accessibility Educational Specific Model

Talal Albalawi[1], **Abdulelah Algosaibi**[1], **and Khalid Aljohani**[1]
[1]Department of Computer Science, Kent State University, Kent, Ohio, USA

Abstract - *Web accessibility is an important topic that web designers and managers take into consideration when developing websites. Originally "web accessibility" referred to helping people with special needs have access to websites; however, now there are sets of guidelines to help make websites more accessible to different kinds of users. Guidelines like the WCAG 2.0 exist to help make the website accessible in a general manner. Having educational websites raises the issue of having these websites accessible by students at different educational levels. Since no educational specific model has existed to evaluate the accessibility of educational websites, we present an educational specific model which we call WAESM. We propose an initial set of guidelines that comprise our model. These guidelines, we believe, should be followed when designing educational websites.*

Keywords: Web metrics, Web accessibility, Web education

1 Introduction

Most of the current educational websites suffer, in terms of accessibility, from problems which prevent the users from fully accessing and thus fully benefiting from such websites. Web accessibility is about making the website accessible for different levels of users and also to people with different levels of abilities and disabilities. Moreover, making the website accessible to all Internet users regardless of the type of the browsing technology they're using is important. In education, web accessibility is about making the websites easy to use for students in terms of easy navigation through the webpages and also about having the websites' content clear for students. With current technologies some websites may not be accessible from some devices, e.g., PDA or touch screen tablets. Web accessibility for educational websites should consider the accessibility issue regarding devices, e.g., text alternatives when a speaker is not available. Another educational concern is avoiding long webpages. There are criteria that should be taken into consideration in order to make a website more accessible. Project managers and evaluators should consider specific features for each given website and the need to customize their criteria guidelines in order to find and address the issues which are important for their websites. There are some organizations that address general web accessibility guidelines; these organizations include the United States Access Board (Section508), the EU Web Accessibility Benchmarking Cluster (WAP Cluster), and Web Content Accessibility Guidelines (WCAG) 2.0.

In this paper WCAG 2.0 is the main starting point that is used to guide the development of our model. A number of the WCAG 2.0 guidelines are, however, not related to our work. Therefore some of these guidelines have been eliminated for our model. An example is "Guideline 1.2 Time-based Media: Provide alternatives for time-based media".

Each WCAG 2.0 guideline has a weight associated with it. These weights are three levels. They are "AAA" which means these guidelines MUST be applied, "AA" means these guidelines SHOULD be applied, and "A" means these guidelines MAY be applied. These weights determine how important the guidelines or criteria are and how closely they should be followed. We combine these criteria into a model, which we call WAESM, to evaluate the accessibility of educational websites; a full description of the model is given in the Evaluation Method Section of this paper.

This paper introduces the WAESM model which is a set of guidelines that help web designers to build accessible educational websites. The paper includes a related work section with an explanation of why we focus on educational websites. After that we present the WAESM model in the Evaluation method. We present the Model testing and result ,and Discussions . Finally, we have the Conclusions and Future work section.

2 Related work

The United States Access Board (Section508) [1], the EU Web Accessibility Benchmarking Cluster (WAP Cluster) [2], and Web Content Accessibility Guidelines (WCAG) 2.0[3] provide sets of guidelines in order to make websites accessible. However, there are no specific educational purpose guidelines. Joe Clark in Building Accessible Websites book [4] explains how to build accessible websites; other talks about Constructing Accessible Web Sites [5] and explain how to create or make a website accessible. But they explain the general guidelines with some techniques to follow and do not mention specific guidelines for educational purposes. Julie A. Smith [6] studied and focused on web users with visual disabilities with no consideration to children's educational websites. Some researchers [7] [8] address issues of web accessibility for disabled people. Some studies [9] [10] [11] address metrics and do surveys about formulas in order to measure the website accessibility for different attributes. Another type of research focuses on the usability of educational websites [12] [13] and gives a statistical analysis. It is based on asking children questions and observing how they use the educational website to find the answers. Our research (WAESM) is mainly focusing on helping to create educational websites that are highly accessible for children in

terms of learning. We present the WAESM model to evaluate the accessibility of educational websites.

2.1 Why educational website

In today's complex world, the level of mastering the basics of reading, math and computers is closely related to children's opportunities to learn. However, class sizes and the quality of teaching in schools are issues that could prevent children from mastering these skills. As a result, the need for other teaching choices to help children to master important skills is growing. Educational websites appear to be one of these choices, and it may in some ways be the best choice. According to NCES [14] (National Center Education Statistics) during the 12-month 2004–05 school year, 37 % of public school districts and 10 % of all public schools nationwide had students enrolled in technology-based distance education courses. This represents an estimated 5,670 public school districts and 9,050 public schools. The number of student enrollments in technology-based distance education courses increased from an estimated 317,070 enrollments in 2002–03 to 506,950 in 2004–05. These statistics show that educational websites are becoming more and more important. Moreover, there are many other good reasons to help grow educational websites. The ease of use of these websites is one. An important factor in the ease of use of educational websites is how accessible the educational websites are. Unfortunately, many educational websites cannot be considered as accessible websites because they do not follow accessibility guidelines that are proposed by organizations such as WCAG 2.0. Also, though WCAG 2.0 guidelines address the general model of website design, it gives no consideration to educational website design. Although it lists some possible guidelines that could be used in an educational model, these guidelines merely scratch the surface in educational website design. The WCAG 2.0 guidelines describe a general method to make the website accessible in general. Some of these guidelines can be applied to educational websites designees but they are not enough. This and many other reasons motivated the research behind the introduction of WAESM. In short, educational websites are growing and getting more and more attention. On the other hand, the accessibility of these websites is not getting enough attention. We think it is important to address this issue.

3 Evaluation method

Web accessibility is an important issue as mentioned in the introduction section of this paper. Standardization for evaluating web accessibility is an important part of the evaluation process. In order to do that there have to be certain standards or guidelines to use. The first set of guidelines we found to be helpful in addressing web accessibility come from the Web Content Accessibility Guidelines (WCAG) 2.0. What WCAG does is provide guidelines that make websites more accessible while taking into consideration people with special needs. Using a number of principles and having a number of guidelines under each principle, each statement in the guidelines is ranked A, AA or AAA. If a certain guideline is given AAA it means that this criterion is of highest importance. These guidelines are statements of design issues which according to WCAG should be present in any website.

For our research, we used their ranking mechanism and some of their principles. Please see Table 1. In addition since the focus of this paper is to test the accessibility of educational website, we consulted some of the educational experts in the field of web education, and some of the guidelines in our model are based on their suggestions. We came up with twenty-three guidelines to evaluate educational websites based on WCAG 2.0 and experts' suggestions. These guidelines are in Table 2. As seen, some of the guidelines remain the same as listed in WCAG 2.0 though we did adjust the rank associated with some.

Table 1: Major WCAG 2.0 Guidelines used in WAESM

Guidelines	Weight
Non-text Content: All non text content that is presented to the user has text alternatives that serve the equivalent purpose.	A
Info and Relationships: Information, structure, and relationships conveyed through presentation can be programmatically determined or are available in text.	A
Meaningful Sequence: When the sequence in which content is presented affects its meaning, a correct reading sequence can be programmatically determined.	A
Use of Color: Color is not used as the only visual means of conveying information, indicating an action, prompting a response, or distinguishing a visual element.	A
Audio Control: If any audio on a webpage plays automatically for more than 3 seconds, either a mechanism is available to pause or stop the audio, or a mechanism is available to control audio volume independently from the overall system volume level.	A
Page Titled: Webpages have titles that describe topic or purpose.	A
Focus Order: If a webpage can be navigated sequentially and the navigation sequences affect meaning or operation, focusable components receive focus in an order that preserves meaning and operability.	A
Location: Information about the user's location within a set of webpages is available.	AAA
Headings and Labels: Headings and labels describe topic or purpose.	AA
Language of Parts: The human language of each passage or phrase in the content can be programmatically determined except for proper names, technical terms, words of indeterminate language, and words or phrases that have become part of the vernacular of the immediately surrounding text.	AA
Unusual Words: A mechanism is available for identifying specific definitions of words or phrases used in an unusual or restricted way, including idioms and jargon.	AAA
Abbreviations: A mechanism for identifying the expanded form or meaning of abbreviations is available.	AAA

Error Identification: If an input error is automatically detected, the item that is in error is identified and the error is described to the user in text.	A
Error Suggestion: If an input error is automatically detected and suggestions for correction are known, then the suggestions are provided to the user, unless it would jeopardize the security or purpose of the content.	AA
Help: Context-sensitive help is available.	AAA

Table 2: WAESM model list of Criteria and associated weights

Guidelines	Weight
Apply effective and simple navigation within the website. •Focus Order •Location	AAA
Create consistent layout of the web pages in terms of color, shape and structure.	AA
Use appropriate font size along with short and simple sentences.	AA
Support the lessons with video and audio content "Multi-mode".	AAA
Apply standard features and functionality, which should be embedded in every web pages of the website depend on the student grade level.	AA
In case of using a questionnaire at the end of the lesson "exit ticket", provide capability to gather a variety of types of data (multiple choice, short answer, and extended response) to provide both students and teachers with feedback on student progress.	AA
Avoid long web pages that make students scroll down and up e.g. a picture should be side to side with its description.	AA
Provide easy search mode that ensure bringing the right result to the student in a simple and easy way.	AA
Provide, for each web page within the website, a unique keyword that helps student to reach a specific lesson.	A
Provide some evidence that the content on the webpage is consistent with best practices in teaching and learning . This is left to the designer to judge since the best practices in education differ from age to age.	AAA
Non-text Content	A
Info and Relationships	A
Meaningful Sequence	A
Use of Color	A
Audio Control	A
Page Titled	A
Headings and Labels	AA
Language of Parts	AA
Unusual Words	AAA
Abbreviations	AAA
Error Identification	A
Error Suggestion	AA
Help	AAA

4 Model testing and results

To evaluate our proposed method (WAESM) we used the manual testing approach. A manual test consists of a set of guideline criteria to be followed in creating a website. It measures a website's educational accessibility for all types of prospective users. The test is done by using a checklist or written evaluation. Some advantages of the manual test are human reviewers can help ensure clarity of language and ease of navigation, and a manual test allows for finding accessibility problems which cannot be found programmatically. For example, is the description provided about an image enough? Therefore a manual based program is developed based on the same principle as in WCAG. In the program the user is given the list of criteria specific to education as listed above in Table 2; for each question the answer is either satisfied or not satisfied. Each criteria has a weight, A= 1, AA = 2, AAA = 3. The highest possible total is 44. We used the following equation to find the percentage of accessibility for each webpage in the website:

$$WebpageAccessibility = \frac{TotalWeightForWebpage}{TotalWeightForTheUsedModel} X100$$

This metric is a general approach to evaluate webpage accessibility. In some research [9], a number of metrics are proposed based on the attributes to evaluate, e.g., blind people's accessibility to the webpage. In our model we want to apply the evaluation formula to test the accessibility level for a given webpage. Thus, this equation is suitable for our purpose. The closest equation to ours is mentioned in [11]; it is called WAB. However, it is based on an automated model which is not fit to be used in our manual model.

According to Bambang Parmanto and Xiaoming Zeng [11] "The metric must be fair by taking into account and adjusting to the size and complexity of the web sites. Web sites may range from a single home page to large corporate sites comprising thousands of pages. A metric that takes into account size and complexity would allow a fair comparison between web sites of various sizes." However, in our model the number of pages is not relevant because the accessibility test is to test for a single page, e.g., a lesson of long division.

For the associated weights, we found that WCAG 2.0 model has a total weight of 117. To evaluate the chosen websites according to WCAG 2.0, we used the services of ACHECKER [15]. This website is a tool that checks a single webpage to see if it follows or is missing any of the requirements of WCAG 2.0. It uses a crawler to visit that page and evaluate it. It generates the results by showing the problems the page has by displaying the list of missing guidelines according to WCAG 2.0 and a list of potential problems, which we don't take into consideration in this study. Next we calculate the website weight and again apply the same equation above to find the percentage of website accessibility.

For our testing purposes to compare WAESM and WCAG 2.0, we choose four websites, three of which are educational websites and one non-educational website. Inside these websites we evaluate a single page at a time. These websites are:

www.kidsnumbers.com
www.coolmath4kids.com
www.funbrain.com
www.cnn.com

The results are displayed in Table 3, which shows the results of evaluation of the list of websites using the WCAG 2.0 and the WAESM model.

Table 3: Result of Evaluation of web accessibility using WCAG 2.0 and WAESM model

Website	Type	WCAG 2.0	WAESM
Kidsnumbers	Educational	%95	%54
Coolmath4kids	Educational	%94	%59
Funbrain	Educational	%93.16	%65
CNN	News	%87.17	%43

5. Discussion

The guidelines listed in Table 1 are general guidelines that can be applied to any website design. What is missing in the WCAG 2.0 guidelines is a focus on educational principles when it comes to website designs related to education. In Table 2 the guidelines are specific to educational websites. Such guidelines enable more accurate assessments of the accessibility for educational purposes of webpages. In this paper, we focus on how accessible a webpage is in terms of reaching the required information in an easy way. In our testing, we used a manual approach to evaluate the websites. Our evaluation verifies that our model places more restrictions when compared to the WCAG 2.0 model, and of course, this was to be expected as we are focusing on educational criteria. As seen in Table 3, the best educational website was funbrain.com which is evaluated by other education experts who place it in the top 20 websites in education. This helps verify that our model works and generates the good results.

In testing CNN.com we wanted to show that since it is not an educational website it would have the lowest accessible rate among all websites we tested, and in fact, because CNN.com violated some of the guidelines in WAESM, it received the lowest rate among all tested websites.

Figure 1 clearly shows that the general WCAG 2.0 guidelines treat educational websites as any other regular website. When it comes to educational websites, the layout, color and organization of the page are important. WAESM highlights them and other educational issues. WCAG 2.0's accessibility rates for the above mentioned websites are high and acceptable. In terms of an educational accessibility rate, these rates are different. The difference is a result of switching from general criteria (WCAG 2.0) to education specific criteria (WAESM). Some criteria's weights in WAESM are more than their weights in WCAG 2.0. Also, some criteria from WCAG 2.0 have been removed due to the irrelevancy. On the other hand, some criteria have been added to WAESM.

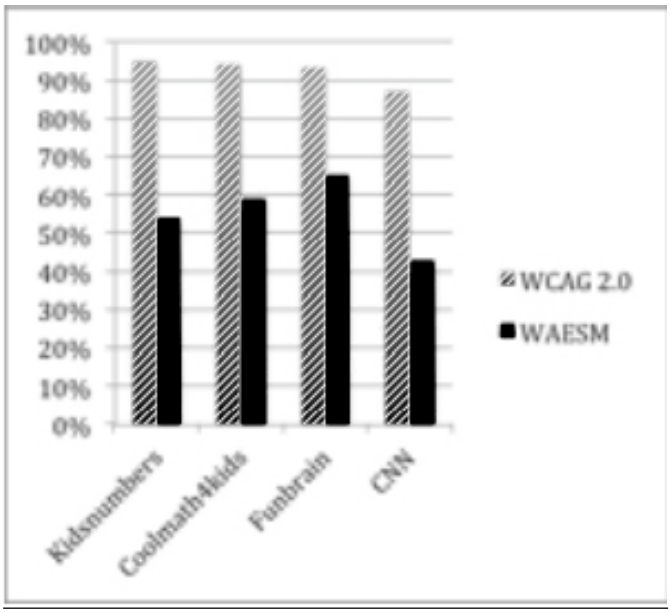


Figure 1: Comparison between the results of WACG 2.0 and WAESM Models

We think that our model produced the expected results. We may need to adjust the weights. Of course, more testing with more websites and more discussions with educational leaders are needed to help refine and improve our model.

6 Conclusions and future work

In this paper we presented the WAESM model, which is a criteria-based model to evaluate and assist in developing educational websites. This model is based on the WCAG 2.0 criteria and the expertise of experts in the educational field. WCAG 2.0 criteria in general do not focus on the needs and purposes of educational websites. We compared the two models, and we showed that the WAESM model does place restrictions on the evaluations of educational websites. It did generate the expected results. Future work includes changing the manual evaluation process in the WAESM model so that it can be done in an automated manner using a crawler and then we can test more websites to see how our model may be refined and improved.

7 Acknowledgments

We would like to thank Professor Austin Melton for providing the guidance throughout the different steps in this study, Professor Mike Mikusa for his insights on what important criteria have to be present in educational web sites, and finally Professor Paul Wang for directing us into the right direction in building our educational model.

8 References

[1] United States Access Board, Section 508 http://www.access-board.gov/508.htm

[2] The EU Web Accessibility Benchmarking Cluster, Evaluation and benchmarking of Accessibility http:// http://www.wabcluster.org/

[3] Web Content Accessibility Guidelines (WCAG) 2.0 http://www.w3.org/TR/WCAG20/

[4] Joe Clark . 2008. Building Accessible Websites. http://joeclark.org/book/

[5] Jim Thatcher, Paul Bohman, Michael Burks, Shawn Lawton, Henry Bob Regan, Sarah Swierenga, Mark D. Urban, Cynthia D. Waddell. Constructing Accessible Websites. 1st edition, April 2002, Peer Information Inc.

[6] Julie A Smith. 2009. Developing Web Accessibility: Section 508 Compliance Of post-Secondary educational Website home pages. Capella University.

[7] R. Lopes, D. Gomes, and L. Carriço. Web not for all: A large scale study of web accessibility. In W4A: 7th ACM International Cross-Disciplinary Conference on Web Accessibility, Raleigh, North Carolina, USA, April 2010. ACM

[8] How People with disabilities access the web http://www.w3.org/WAI/EO/Drafts/PWD-Use-Web/

[9] A. P. Freire, R. P. M. Fortes, M. A. S. Turine, and D. M. B. Paiva. An evaluation of web accessibilitymetrics based on their attributes. In SIGDOC '08:Proceedings of the 26th annual ACM international conference on Design of communication, pages (73,80), New York, NY, USA, 2008. ACM.

[10] Devanshu Dhyani, Wee Keong Ng, and Sourav S. Bhowmick, A survey of web metrics, ACM computer survey, Vol 34, No. 4. December 2002, pp. (469,503).

[11] Parmanto, B. and Zeng, X. Metric for Web Accessibility Evaluation. submitted to Journal of Society for Information Systems, 2003.

[12] Shiva Naidu. July 2005. Evaluating the Usability of Educational Websites for Children. Usability News July 2005, *Vol. 7 Issue 2*

[13] NIELSEN, J. (2002, April 14). Kid's corner: Website usability for children. Jakob Nielsen's Alertbox. Retrieved March 10, 2005, from www.useit.com/alertbox/20020414.html

[14] National Center for Education Statistics. http://nces.ed.gov/fastfacts/display.asp?id=79

[15] ACHECKER, Web Accessibility Checker. http://achecker.ca/checker/index.php

Best Practices for Project Management: A Further Study

Kathleen Stirbens, Rafael Feijo
Department of Computer Science, Kent State University, Kent, Ohio, USA

Abstract *- Project managers have many challenges to overcome during all phases of a project. Choosing the right tools and metrics can make or break the project. This paper describes three guidelines, which when taken into consideration, have led projects to success. Metrics useful in management are also discussed as being very useful and applicable to IT development. Moreover, a new guideline was obtained by interviews with project managers of XYZ Company.[1] This new guideline helps provide good estimates of time and development at the beginning of a project. We will also be introducing the ideas of cohesion and coupling in the context of complex systems consisting of multiple programs.*

Keywords: Software metrics, developer skills, error detection, project management, cohesion, coupling.

1 Introduction

Several researchers have tried to define the basic methodology which goes into creating a successful project. There has not been an overarching methodology developed which can apply to each and every project. In this paper, we do not promote an explicit overarching methodology, but we do present four guidelines which when incorporated into a development methodology can significantly improve the likelihood of successful projects.

In this paper, successful completion of a project is defined as the project being released on time, being free (completely free or "free" according to some acceptable standards) of faults in the system (commonly referred to as "bugs"), and being within the budgeted cost parameters. The current research shows that developer skills, detection of faults and complexity of the product to be developed are all critically important factors which must be considered by software product managers if products are to be completed successfully. Our guidelines which will be discussed in detail in this paper directly relate to these important factors.

Research has shown that certain general management metrics are useful and applicable to IT management situations. A discussion of these metrics will show how their use may produce similar positive outcomes for a variety of projects including IT/IS projects.

In addition, interviews with project managers conducted for this research have yielded another factor which should be an integral part of the software management process. This factor looks at previous projects to estimate different aspects of the new project. Past experience can serve as a basis for the estimates needed for the new project. While this may not be a part of the actual development process, it can be extremely useful to the successful completion of the project.

2 Existing Guideline #1 Developer Skills

The first important attribute of the programming process which must be analyzed when management plans an IT project is the skills of the persons assigned to the project. Companies try to hire the best employees, but no employee possesses the entire knowledge of every system and every situation. Even the most competent programmer has a limited set of skills to bring to their work. In addition, employees who are newly graduated from college can lack experience with a variety of products and programs the company markets and sells.

Developers' lack of experience in programming can create havoc throughout the system. From a lack of understanding of the system's essential design, to a missed issue leading to vast amounts of time and cost to repair. The consequences of this lack of experience can be devastating to the successful completion of a software project.

The response to this dilemma can be equally frustrating for managers and job seekers alike. Managers argue that they need to use inexperienced programmers in order to eventually produce experienced programmers. Job seekers, on the other hand, are told they need to have experience for the job, but they need jobs to gain experience.

Perhaps mentoring programs can help to alleviate this disparity. Pairing newly hired or newly graduated programmers with seasoned employees may help to build the skills necessary for the development process. New graduates can gain useful knowledge acquired by years of experience.

The inexperienced programmer could perform important tasks that build her/his skill set under the guidance of the experienced developer. In addition, seasoned employees may acquire new skills and fresh techniques.

[1] The company's actual name is not being used.

3 Existing Guidedline #2 Detection of Faults in the Program

Ordonez & Haddad [8] have researched the state of metrics in the software industry. One metric that Ordonez & Haddad [8] researched is the cost of defects or faults in the program. Better known as bugs, these problems can cause massive cost overruns depending on the stage of the software development cycle in which they are detected.

Ward in the Hewlett-Packard Journal has developed a computation to determine "the real cost of software defects." [12] HP's Waltham Division maintains a reliable database of software quality metrics which Ward used to develop the computations. This database contains logs of defects in the software, product comparison studies, post-mortem studies, code complexity and size analysis, and resource plans from a variety of phases in the software development cycle.

HP has found that the effort to find and fix one typical software defect is 20 hours. If we use $75 per hour as a typical expense for a software engineer, one defect will cost us $1500 to fix according to the metric:

Rework cost per software defect = 20 hours x $75/hr = $1500.

From interviews we conducted with project managers at the XYZ Company, there is a generally accepted maxim/theory/rule of thumb which states that bugs and errors caught during the development phase will cost the company a unit of cost to correct. If the bug were to be caught later in the testing phase, it will cost 10 times the original unit cost. If the error is caught much later, once the product has been released to the end user or customer, it will cost 10 times more than if it was caught in the previous phase, making it 100 times the original cost if found in the beginning phase.

If we use the work of Ward and HP as a basis for our calculations, we have an average cost of $1500 for one defect to be found and corrected. This is an average across all testing phases. [12] If 20 hours is an accurate estimate of time needed to fix one error, the results can be staggering.
To find and fix this in the testing phase, the dollar cost of these defects can then be calculated as:

Software Defect rework cost = 20 hours x $750/hr = $15,000.

Finding the same defect in the released version of the software would bring the cost to:

Software Defect rework cost = 20 hours x $7500/hr = $150,000.

Regardless of when the error is caught, the developers must look to solve these problems. The product must be reworked, rebuilt, and sometimes rereleased. Many of these costs cannot be passed on to the customer, so the company must assume the additional expenses.

In addition, these defects in the product can delay the release of the product. This causes cost overruns or even lost revenue from promised deadlines being missed.

In order to have the least amount of bugs in an application being developed, the project manager should take into consideration two factors: unit test plans and third party dependencies. The project manager should also allocate extra time to the project when using third party dependencies. This accounts for risk of bugs being introduced by the third party application. Issues related to third party software are also discussed later when we develop an expanded scope and applications for cohesion.

Errors and bugs require more time and more cost when they are found and fixed later in the process. It is helpful to find and fix these early in the life of the project. This might be a way to utilize the existing skills of the inexperienced programmers. By setting the inexperienced programmers to the tasks of early and pre-release programming, they can use skills they already possess and develop new skills that they are acquiring in the process.

4 Existing Guideline #3 Management Metrics for IT/IS

Cooke-Davies [4] delineates the "real" factors that lead to success on projects. It is suggested that there are 12 factors critical to the management of the project, the success of the individual project, and consistently successful projects. In research done in 1993, the research company named Human Systems became the first research company to identify "project management best practice." (sic)

Formed by 15 European companies, this company has expanded to include 70 organizations throughout the world. They research methods to improve project performance. This research dealt with a variety of project management environments such as engineering, defense, petrochemical exploration, construction, pharmaceuticals, and IT/IS systems integration. While not all these projects were in the area of software engineering, they all have similar outcomes when they are successful, and we believe many of these factors can contribute to successful IT/IS projects. Cooke-Davies states that each successful project can contribute to the corporation's value. Successful projects can be a measure of corporate success. [4]

Statistical analyses of national and multinational companies have yielded factors that Human Systems considers important in the three areas important to project success. Our work in this section on developer skills draws heavily on the research done by Cooke-Davies based on the research used in his

submission of a dissertation to Leeds Metropolitan University. [5]

It is noted, however, that the "human factors" involved are quite often overlooked in examining success factors. Cooke-Davies [4] states that "it is fast becoming accepted wisdom that it is people who deliver projects, not processes and systems." Though this is not a new idea in software development, it is certainly worth emphasizing.

Lechler [6] emphasizes this point in the title of his paper; the first part of the title is "When It Comes To Project Management, It's The People That Count."

According to the project being developed, it makes sense to choose a developer who is familiar with the product being developed. Umarji & Seaman [10] found that managers find great diversity in the extent of expertise in the teams that they manage. Developers range from novices with recent degrees to very highly experienced senior level programmers.

In their research, they had developers answer questions regarding the usefulness of metrics for project management. The developers ranged from senior level to recent college graduates. The purpose of this research was to validate whether metrics from developers were accurate and useful to the project. The metrics used were usefulness, intention, attitude, and ease of use of metric tools. This is important since developers must specify how long they took to produce a feature, which is then used by the project manager as a metric. Giving wrong numbers can cause a well planned project to look like a failure at any point in time. [10]

When it comes to reporting their time for a feature, developers may think twice about the number they provide, since it can be used against them. "The findings indicate that developers perceive metrics as a threat – they believe that their performance might be monitored by metrics and that failure to comply with the business goals would impact their career goals." [10]

Another reason for not reporting accurate metrics was the difficulty of using or learning to use the metric tools. "They were neutral about the tool being easy to use and well-integrated, but they felt that learning to use the tool had been tedious and even after having a tool, it took too much time to report measure." [10] Most of the metric tools require the developer to itemize their work. This means that for each little task, the developer had to create an entry form describing the task, problem encountered and solution taken, as well as time spent. When developers are overloaded with multiple tasks and are pressured to complete them in order to meet deadlines, they are not very likely to verify that their metrics on each task were accurate. This leads developers wondering what is more important, the actual work to be delivered or the numbers spent on the work so that they can be presented to a manager. This situation is far too common in all companies.

The last finding from Umarji & Seaman's research [10] was the correlation of low expertise with low confidence, "While we did not have a specific question on metrics expertise and knowledge possessed by developers, one of the immediate side effects of low expertise is low confidence in the ability to perform a task i.e. low self-efficacy." [10] It is important for project managers to assign young developers a lighter load, so that they do not feel overwhelmed. Otherwise, they can feel stressed, report very inaccurate metrics, and slow the progress of the project. The lack of experience of the developer can affect the project in a myriad of ways, including delays in time, and more bugs in the program at various levels. This translates directly into cost overruns and missed deadlines.

Umarji & Shull [11] have examined product metrics which measure customer satisfaction with the finished product. In addition, they have delved into process metrics which measure the effort and effectiveness of the development process. Their research has found that there are problems in using metrics to measure the results of a project. While the metrics themselves may not contain inherent problems, the way the metrics are perceived and utilized may cause the problems.

When metrics are used to measure product or process, the developers of the software may perceive them as problematic. Developers may perceive these metrics as unfair, and view them as a threat to their advancement of their careers. When metrics are thought of as unfair, developers take measures to protect themselves from the consequences of these applied metrics. When the perception exists that careers are threatened, people will make sure that the numbers reflect a brighter picture than exists. This may render the metric useless as an accurate measure of anything.

5 Existing Guideline #4 Complexity of the Software

Software Complexity (SC) is determined by using a variety of metrics to measure the software. It can be as direct a metric as Lines of Code to measure the size, or as complicated as cohesion and coupling to describe the amount of integration within the program.

Lines of code (LOC), number of modules (NOM), number of functions (NOF), number of local functions (NLF), number of public variables (NPV) , and number of public functions (NPF) can be seen as metrics which measure attributes within a specific program. These are well known metrics to measure size and structure of programs.

Two more criteria named cohesion and coupling are important software measurements. Cohesion is more specifically aimed at a single module. Cohesion is defining how encapsulated the module in a program is. Does it cover

more than one topic? Or is it highly defined? The more cohesion there is, the more self-contained the module is.

Booch and others [1] state that cohesion "comes from structured design. Simply stated, cohesion simply measures the degree of connectivity among the elements in a single module (and for object-oriented design, a single class or object)." The best form of cohesion is functional cohesion. If we have a class in programming called Dog, it is functionally cohesive if "its semantics embrace the behavior of the dog, the whole dog, and nothing but the dog." [1]

Coupling is defining how closely connected modules are to each other. Stevens, Meyers and Constantine speak of coupling as "the measure of the strength of association established by a connection from one module to another." [9] Complexity of the system increases when there is strong coupling. The more coupled the modules are, the more effort it takes to make changes, since the change in one module affects other modules. The programming is more difficult to understand, and harder to program since there is an interdependence from one module to another.

In many software development projects, the scope of the coupling-like connections could be expanded to include other programs and the modules contained in them. The program being developed may need to import output from external programs as the input to its functioning. In addition, the output from this program may be used to interact with another program or programs outside of the system.

Our research suggests that an expanded definition and scope of coupling would enable the project manager to incorporate good programming ideas and practices to more than programs being coded. Many times the intricacies of the project must include other programs with which the project interacts. Previous programs within the company or programs from outside vendors often interact with newly developed programs. This suggests that the definitions of cohesion and coupling could be meaningfully expanded to include these outside programs within the "system" which the manager is creating. The system then becomes the entire set of interacting programs which may share among other things input and output.

The more external dependencies a program has, the higher the complexity the programming is with respect to the enlarged scope of the definition. When there is increased complexity due to this enlarged scope, the coupling ideas take on expanded importance.

Briand, Morasca, and Basili define a system as a collection of framework modules, stating that "all systems can theoretically be decomposed into modules. The definition of a module for a particular measure in a specific context is just a matter of convenience and programming environment (e.g., language) constraints". [2]

We propose that the idea of modularity be redefined and expanded to include multiple programs within a system. If one defines the system as a set of programs, and collections of programs as the modules within this system, then the same metrics which Briand, Morasca, and Basili set forth can be applied to systems of programs.

If the modules are redefined to be programs that are interconnected in a larger system, the concept of cohesion takes on a similar yet expanded meaning. The cohesion now represents how self-contained a collection of programs in the larger system is. Does it have input from another module (i.e., collection of programs)? Does the output from this module go to another module as input? Of course, in some cases we may want to consider a single program to be a module.

We modify the representation of Briand, Morasca, and Basili. [2] The system S = <E,R> is represented by elements and relationships in the system where E is the set of programs in the system. R is the binary relation on E, i.e., R is the set of relationships between S's programs. E is the set of programs, and R is the set of the inputs and outputs shared or passed among programs.

The coupling now represents not connectedness within the programs, but connectedness between them. The same metrics used to describe the workings within a program now reflect the interconnections between programs in the system. In this circumstance, coupling also refers to connections with and interactions with modules from external programs, whether utilizing data from another program or providing data to another program.

Chidamber and Kemerer [3] speak of coupling in object oriented programming. They specify a definition for coupling between classes (CBO) as the number of classes to which a specific class is coupled. This coupling is defined as one object acting on another one, as when one object uses variables from another.

If we modify the concept and theory to refer to programs instead of classes, the concept and theory still hold true.

Three viewpoints put forth by Chidamber and Kemerer express the dynamics of object oriented programming. These are applicable in our modified understanding of coupling.

5.1 Viewpoints [2]

Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application.

This viewpoint of Chidamber and Kemerer is still applicable if we are referring to programs. The more

independent each program is, the easier to use it in another application.

In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult.

This too is applicable to programs in a system. The more variables that two programs share, the more difficult it is to change one program without affecting the other programs in the system.

A measure of coupling is useful to determine the complexity of testing of various parts of a design. The higher the inter-object class coupling, the more rigorous the testing needs to be.

The same reasoning applies here to the programs in a system. A measure of coupling among programs in a system would be helpful in determining the complexity of testing of the various parts of a system's design. In order to accommodate more coupling among programs, more extensive testing is required. The more variables shared among programs in a system, the more coupling exists and the more coupling, the more testing.

These new and modified perspectives regarding cohesion and coupling highlight the importance of inter-program interactions on the cost, the time constraints, the possibility of defaults in the programming, and even whether deadlines are met in an on-time manner when designing and developing large systems. Keeping cohesion in programs at a maximum and the coupling among these programs at a minimum will result in the cost of implementing the entire system being minimized. Further studies may be able to determine if different guidelines are necessary for this expanded definition of cohesion.

6 New Proposed Guideline – Utilizing Past Software Projects

The project managers interviewed stated that in order to have a successful software project, one should, when possible, look to past similar software projects to base the estimation for the current project. This allows the project manager a good starting base from which to launch the new project. When existing software projects are similar to the new needed project, using data about the earlier software projects can streamline and improve the process of estimating cost and time for the new project.

Based on our interviews, the main points to base this estimation of cost and time for the new project on are number of defects, time spent by previous developers, and determining future dependencies on third party software of the current project. From these three of points, a manager can expect to allocate X amount of time for the development of the new software as well as having to deal with Y number of roadblocks from bugs and third party dependencies.

The number of defects and the complexity of the defects from past projects can help determine issues which the current project must address. Past errors found can help to delineate the types of errors which can be avoided in this project. Taking this a step further, as the software project progresses, a database of bugs (either fixed or existing) can be put together, so that in turn, it can be useful for another similar software project in the future. By keeping these data, project managers can begin to move from using past experiences as intuitive guidelines to developing simple and them more complex metrics for predicting future needs. This is, in fact, how new metrics can be developed.

Knowing how long a specific feature took to develop in past projects can give one a sense of the time required to produce a similar feature or one with added functionality in the current project. Past experience might actually decrease the amount of time needed to replicate a similar feature. Further studies may be able to create metrics to determine the time savings for subsequent systems.

In order to have a starting estimate of time and resources for a project, a project manager should start by looking at previous similar projects. Factors to look for in previous projects are number of bugs, total time spent, and third party dependencies. By looking at these factors, a manager can know what kinds of bugs may need to be fixed as well as how many bugs to expect. A manager can also estimate how long the new features of the new software will take to develop. Finally, a manager can predict what external dependencies the new software will require and so, plan extra time for integration.

7 Conclusion

This research has examined the current thinking which determines that developer skills, detection of faults and complexity of the product to be developed are all integral parts of the software product management. When applied appropriately, metrics can help lead to successful projects. However, project managers should be aware of potential personnel issues involved in implementing a successful metrics program. Further, by modifying the work of Meirelles and Chidamber and Briand et al we have expanded their theories and models to include the concepts of metrics for cohesion and coupling of large multi-program systems.

While there are metrics to measure many aspects of programs, there are few that directly or indirectly address the skill levels of the human portion of the equation. Most metrics do not factor in the people, and yet it is generally recognized

that this is a very important part of the project management. Future work could include data collection which would allow the managers to more precisely assess the skills of the people they manage. There are issues as stated by Umarji & Seaman which may limit the veracity and usability of such data.

The successful completion of the software project is directly impacted by the complexity of the software. While there are many metrics which measure various attributes which contribute to complexity, in this research we explicitly examined the two attributes coupling and cohesion. These attributes are usually defined within a program, and even are many times limited to the modules being measured. Our research found that the scope of these two attributes could be meaningfully expanded to include company programs and third party programs with which the current project must interact. This expanded scope would increase the complexity of the metrics program, but also could help ensure the successful completion of the project.

The managers who were interviewed utilize past experience to estimate, verify, and extrapolate the possibilities of success for the new project. Past experience can reveal many patterns and trends which can illuminate the current project. Timelines, defects found, experience of the programmers and interdependency can all be gauged from past projects and the metrics used to evaluate past performance.

We suggest a research program that would use data from already completed projects to estimate data for new projects and then would compare the estimated new project data with actual new project data so that we could begin accurately and statistically to use completed projects to plan for and develop new projects. We would then be able to define estimation or prediction metrics which would use similar completed projects to produce estimates for new and developing projects.

8 Acknowledgements

We would like to give thanks to Dr. Austin Melton, Computer Science professor at Kent State University, for his direction and pointers that provided us with new ideas in which to extend our research.

We would also like to thank M. Brown for his insightful perspective as a project manager.

Finally, we would like to thank the class of Software Metrics at Kent State University for providing feedback to our ideas for this paper.

9 References

[1] G. Booch et al., "Classes and Objects," in *Object-Oriented Analysis and Design with Applications*, 3rd ed. Redwood City, CA, Addison Wesley Longman Publishing Co., Inc., 2004, ch. 3, sec. I, p. 113.

[2] L. Briand et al., "Property-Based Software Engineering Measurement," *IEEE Trans. Softw. Eng*., vol. 22, no. 1, pp. 68-86, Jan. 1996.

[3] S. Chidamber and C. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Trans. Softw. Eng.*, vol. 20, no. 6, pp. 476-493, June 1994.

[4] T. Cooke-Davies [2002], "The "real" success factors on projects," *Int. J. of Project Manage.*, vol. 20, no. 3, pp. 185–190, Mar 2002.

[5] T. Cooke-Davies [2000], "Towards improved project management practice: Uncovering the evidence for effective practices through empirical research," Ph.D. dissertation, Leeds Metropolitan University, Leeds, UK, Aug 2000.

[6] T. Lechler, "When it comes to project management, it's the people that matter: an empirical analysis of project management in Germany," Int. J. of Project Manage., vol. 20, no. 3, pp. 185-190, Apr 2002.

[7] P. Meirelles et al., "A Study of the Relationships between Source Code Metrics and Attractiveness in Free Software Projects," in *2010 Brazilian Symposium on Software Engineering*, Salvador, Brazil, pp.11-20, 2010.

[8] M. Ordonez and H. Haddad, "The State of Metrics in Software Industry," in: *ITNG '08: Proceedings of the Fifth International Conference on Information Technology: New Generations*. Washington, DC, IEEE Computer Society, pp. 453-458, 2008.

[9] W. Stevens et al., "Structured Design," *IBM Systems Journal*, vol. 13, pp. 60-73, 1974.

[10] M. Umarji and C. Seaman, "Gauging acceptance of software metrics: Comparing perspectives of managers and developers," in *3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 236-247, 2009.

[11] M. Umarji and F. Shull, "Measuring Developers: Aligning Perspectives and Other Best Practices," IEEE Software, vol. 26, no. 6, pp. 92-94, Nov./Dec. 2009.

[12] W. Ward, "Calculating the 'Real' Cost of Software Defects," Hewlett-Packard Journal, 42(4), pp. 55-59, Oct. 1991.

Improving Organizational Performance Using Academic Assessment Techniques

Binamra Dutta
Hewlett Packard, Vancouver, WA, USA

In the past few decades, the human resource departments have started utilizing metrics to gauge the performance of employees and to make sure that the human capital is aligned properly with the business goals as well as with the knowledge and skill requirements for their respective roles. While beneficial if applied correctly, human resource metrics focused on accessing employee performance and skills also have the potential for creating a stressful and unproductive environment for employees. This paper analyzes the challenges related to the human aspect of such performance metrics programs and suggests ways of successfully applying classroom assessment techniques to human resource performance assessment programs for motivating the people involved.

1 Introduction

As the economy has transitioned from being physical production oriented to service provision oriented it has become important to develop human capital performance metrics that not only judge individual performance, but also correlate it with the achievement of broader corporate objectives. In current times, human capital has become more important than physical capital that has dominated industry in the past. As there is no one trait to assess an individual's performance, a lot of factors like job specific skills, individual goals, teamwork skills, organizational compatibility, motivation and future growth potential have to be kept in mind while developing human resource measurement and metrics.

An important factor to consider when developing a human resource metrics program is the challenge of collecting 'invisible' data. This refers to the human aspect of metrics program and the various attributes of a human being which are always difficult to collect. Some metrics may also result in 'measurement dysfunction', in which participants alter their behavior to optimize something that is being measured instead of focusing on the real organizational goals. As an example, if an organization is measuring productivity based on lines of code produced by a programmer as opposed to quality of code, it is expected that some developers may change their programming style to visually expand the volume of code they produce or code quickly without regard for defects.

2 Human Resource Development Metrics, Its Challenges and Recommendations

In the present day, globally-competitive, knowledge-based, economic paradigm, people working in a company are the intellectual assets which have come to define how a company performs. The human resource department is responsible for managing this intellectual asset of a company. This responsibility includes having an oversight of a range of activities that should span the entire lifecycle of an individual's employment with the company such as recruitment of new hires, creation of new roles, tracking of applicants, new hire orientations, management of employee benefit programs, employee performance tracking, tracking of compensation, employee satisfaction surveys and employee exit processing. Traditionally, such personnel departments have been highly process oriented and involved for the most part in the creation and management of documentation. Every step from hiring an employee to termination and everything in between had to be carefully documented. With the advent of technology this kind of cataloging and tracking can be easily accomplished using automated tools, allowing personnel departments to focus more on developing ways to better and more effectively utilize the human capital. Using employee performance assessment metrics has been one such initiative. Skill assessment tests, in person interviews, human resource scorecards, human resource programs to enhance attitudes and skills, performance reviews and employee satisfac-

tion surveys are just some of the assessment tools that have now become commonplace.

Although the dynamics of the human capital will keep evolving, employee performance and organizational success will always be linked. The ability to maximize the performance and productivity of the human assets of a company relies on the ability to accurately measure their skills and examine their weaknesses. The challenge in developing a successful performance assessment program is to collect the 'invisible data' meaning the hundreds of discrete human attributes and behaviors that are needed to fulfill a specific role in a company. If employees see the assessment of their performance as unrelated to the job requirements and role expectations, the assessment just becomes an external judgment which doesn't help and might even do a lot of damage to their productivity. This is also the case if they have little or no say in the whole evaluation process and are unclear about the performance criteria. Too often such initiatives and assessments are just seen by employees as another tool for management to control them. The threat and anxiety of an evaluation can prevent an employee from becoming engaged in a project and can dampen the enthusiasm needed for successful projects. In contrast a successful employee assessment program can supplement employee achievement and enhance productivity. If employees can see value in each aspect of their performance assessment and they know that they will receive useful and constructive feedback, helping them grow within the organization then they are more likely to engage positively in the performance evaluation.

For identifying the attributes and traits needed to efficiently perform a job a multi method approach involving various tools has to be put in place. These should include job observation, reviewing existing documentation regarding job requirements, job training materials, structured questionnaires and interviews with the immediate managers to understand the business requirements of a particular job. Once the job requirements, attributes and behaviors have been identified a sound measurement system can then provide the data needed to explore and determine the relationship between these values and the achievement of business goals. This provides a standardized method for rating on the job behaviors like quality, skills, knowledge, accountability and flexibility.

For comparing and contrasting, behaviors have to be organized into ranges of effectiveness or proficiency and ranked in terms of importance to job performance. Standardized assessments are needed for rating an employee on attributes like work quality, dependability, initiative, flexibility, skill building, job knowledge, punctuality and supervisory ability. Such assessments should be designed to accurately collect as much data on behaviors and job outcomes as possible. Ultimately the assessment has to reflect the breadth of the business requirements from a specific role. Rather than having a standard assessment for every employee, an assessment should be tailored for specific categories of employees such as part time, full time, exempt, non exempt, new hires, interns, temporary consultants and contractors. Self and peer assessments can also be used to promote formative activities among employees allowing them to become part of the performance evaluation process. The peer assessment activities should be constructive and promote increased participation within the team, resulting in constructive and valuable feedback.

Formative feedback has to be gathered from employees and managers at various levels on the effectiveness of such assessments and necessary adjustments which might be needed over time have to be made. Continuous monitoring and analysis of the collected data should also be done to determine the effects of the intervention. The leadership vision and strategy for implementing the metrics program should also be communicated clearly at all levels within the organization. This would also permit a more accurate analysis of the correlation between each individual performance and overall corporate performance. Using proper feedback mechanisms all employees should be able to recognize how they can directly influence the measurement results. With proper employee buy-in into the assessment metrics program the employees will push to achieve the desired measurement goals. It may also be desirable to couple the performance metrics program to an awards program for recognizing excellence.

We feel that the biggest challenge in implementing any individual and company-wide performance initiative is to lay the framework based on the missions and objectives of the organization. In short, the challenge is to efficiently collect and analyze individual human attributes and behaviors and then develop a performance assessment initiative for Organizational Performance Improvement.

3 Classroom Assessment Initiatives and Techniques

Traditionally, schools have used grade assessments with the belief that maximizing anxiety will fuel growth in learning. The general idea behind standardized testing in schools was to make the schools accountable to the public as well as to make sure that student's knowledge and skill were measurable on a standard scale. A vast amount of research has been done in this area which has since suggested

that if applied incorrectly there can be a lot negative consequences of classroom assessment. In recent years a shift has occurred in schools to motivate students to become more competent and using assessment as a power tool as opposed to just a 'ranking scale'. We believe that the challenges in applying successful assessment techniques in academics and promoting motivational learning in students are similar to the ones in applying a successful human resource performance assessment program in a company and motivating the people involved. In our review of the existing literature on classroom assessment [1, 2, 3, 4] we found that there are some well defined techniques in performing classroom assessment which can be applied while designing an 'Individual and Organizational Performance Improvement' initiative for an organization.

3.1 Pre-Assessment and Curriculum Planning

Pre-assessment techniques in education are used for measuring the current skill level of students as well as for assessing learning in progress. For example a language teacher may create pre-assessments tools like questionnaires or peer group discussions to find out the current skill level of students in a class. Then after analyzing the assessment results the teacher can then create a tailored curriculum focused on honing specific language skills like grammar, vocabulary etc. which need further improvement.

Similar pre-assessment techniques can be used by companies to determine the current skill level of employees and develop tailored employee training programs based on the assessment results. Various assessment techniques like direct observation, individual / group questionnaires, one to one meetings, focus groups and tests can be designed for such initiatives. To get a complete picture a combination of such tools is required. While designing such assessments, organizational goals and various internal and external requirements should be kept in mind. Careful analysis of the results might point to setting up business process oriented training programs or employee software certifications on specific vendor products. Such training programs can then be included as part of the annual learning goals in the employee performance appraisal.

As part of the learning goal setting process employee feedback can be gathered on which software certification are most valuable for the individual. If certain knowledge areas needing improvement are identified in an individual then adequate steps should be taken for developing those skills. This can include conference registration, formal training, funding for advanced education, self directed training or on the job training. If the issues identified are more systematic, then organization development initiatives like re-organization of teams, team building programs or strategic policy changes can also be put in place.

3.2 Assessment for Accountability

This form of assessment is usually done at the end of a specified period of time to evaluate how the acquired skills after taking a class stack up against standardized educational assessments. For example an end of unit test, final course grades and state board examinations are all standardized assessment techniques used for external accountability. Course grades help in accounting student achievement for those beyond the classroom. Similarly, state exams help in accounting student achievement for guardians, policymakers and college admission officials.

If training programs or organizational changes were put in place as part of pre-assessment planning stage then post-assessment analysis is equally important to evaluate the effectiveness of those programs. Learning can typically be judged by collecting pre-test and post-test assessments and comparing the changes. While learning goals can be easily judged by post assessment tests or passing vendor software certification exams, it is harder to gauge individual feelings about the program and individual post assessment behavior changes. Formative feedback mechanisms should also be put in place to gather a consensus on the benefits of such activities from the participating individuals. The feedback mechanisms should be designed to collect behavioral data which should be further analyzed to suggest changes to the programs. If a program is not deemed as beneficial, then the organization should stop investing in that development activity.

As knowledge and skill development activities may not produce immediate and visible return on investment, there might be a pushback from the executive management for supporting such initiatives. However, job specific training might be accompanied with expectations for immediate performance improvement. The human resource department should make sure that executive management has reasonable expectations for any specific skill development initiative. Applied correctly, such skill development initiatives will in time increase the competence and expertise of employees leading to improved individual and organizational performance.

3.3 Selection and Ranking

Another common use of assessment techniques in academics is for selection and ranking of students among peers or across local and national institutions. Tests are given to students to judge how they differ from each other. These tests generally referred to as Norm Referenced Measures lay more emphasis on reliability of data rather than meeting the curriculum standards of a grade. The underlying assumption for doing this is that some students are smarter than others and grades should reflect differences in ability. Reliability is of primary importance as based on the test data decisions are taken on whether specific students need special programs or whether they should be awarded merit based scholarships etc.

Identifying important attributes and behaviors using Norm Referenced tests can help a company streamline the recruitment process by evaluating and selecting only those candidates which reflect those attributes. Such tests should align with the job requirements and be able to determine individual performance in comparison to the group. The questions should be formulated by various subject matter experts within the company. New questions should be pretested in actual testing conditions and analyzed for potential weaknesses. The questions that are consistently answered correctly by those scoring high on a test but incorrectly by those scoring low on a test can then be included in the standard pool of questions organized by job groups. In some cases, "high" and "low" scores may be determined by the actual job performance of individuals who are hired.

4 Conclusions

Establishing a successful human resource measurement and metrics program in an organization is no small endeavor. Development and implementation of any metrics program involves paying close attention to several categories of critical success factors as discussed in the previous sections. Furthermore, many psychological and personnel issues have to be addressed to increase chances of success from such a program. In this paper, we have provided practical guidelines to the steps we believe are crucial for achieving such a measurement process. We have also discussed recent initiatives in the field of classroom assessment and how those techniques can be successfully applied to motivate employees resulting in increased individual and organizational performance.

5 References

[1] Black P (1999). "Assessment learning theories and testing system"; London, Open University Press, 1999

[2] Assessment Reform Group. "Assessment for Learning: beyond the black box"; Page Numbers (31-57), Cambridge, University of Cambridge, 1999

[3] William D. and Black P. "Meanings and consequences: a basis for distinguishing formative and summative functions of assessment?"; Page Numbers (537-548), British Educational Research Journal, 1996

[4] Pelligrino J. W., Chudowsky N. and Glaser R. "Knowing what Students Know – The Science and Design of Educational Assesssment"; Page Numbers (221-260), Washington DC, National Academic Press, 2001

Extended Cyclomatic Complexity Metric for XML Schemas

Reem Alshahrani
Department of Computer Science, Kent State University, Kent, Ohio, USA
Ministry of Higher Education, Riyadh, Saudi Arabia

Abstract- *Extensible Markup Language (XML) plays an extremely important role in the Web application development process. It is a universal format for data and has been adopted for exchanging information among distributed applications. Because XML documents are usually large, it is necessary to find ways to enhance their ease of use and maintainability by keeping their complexity low. In this research, we propose a revised and improved complexity metric for XML Schema documents (XSD) based on the McCabe complexity metric, which is used to measure cyclomatic complexity. The proposed metric measures complexity according to the complexity of the usage of the XSD components and for binding the XML data into native language.*

1 Introduction

Accessing the Internet is very important to most people. Web usage continues to grow at an astounding rate, and the amount of data conveyed by the many documents available on the Web has also increased exponentially. While information sharing among different parties connected to the Internet has become pervasive, most companies have started providing Web applications to increase business productivity by making data continuously available. The need for information transmission from one party to another has resulted in the emergence of a standard mechanism for information exchange that can be easily implemented by different domains. Extensible Markup Language (XML) is a meta markup language that was developed by the World Wide Web Consortium. The main purpose of XML is to simplify Standard Generalized Markup Language (SGML) by focusing on a particular problem — documents on the Internet. That makes using SGML easier and more straightforward, which makes XML a suitable choice to define document types, to author and manage SGML-defined documents, and to transmit and share documents across the Web. XML has eliminated many of the more complex and human-oriented features of SGML to simplify implementation environments, such as documents and publications. It provides a good combination of simplicity and flexibility and has been rapidly adopted for many other uses.

As information sharing in distributed computing environments becomes more widespread, XML becomes more popular as a universal data format for exchanging structured information between distributed Web applications. Because of its high-performance, XML has been gaining general acceptance as a standard for data representation and transmission between distinct applications running on different networks, such as educational services, digital libraries, and e-commerce. Due to the powerful expressive capability of data and documents and XML's flexible nature and ease of use, it has been adopted as a standard mechanism for structuring data and creating effective information retrieval.

Intense competition between companies to optimize Web application performance for end-users naturally leads to concerns of quality for these applications. XML metrics can be important for predicting the quality and complexity of Web applications. These metrics play a role in ensuring high-performance for XML documents; their complexity can be determined based on various syntactic and structural aspects. Many researchers have started to define complexity metrics for XML documents. In their research to investigate the complexity of an XML Schema, Lammel, Kitsis, and Remy [6] used several well-known procedural metrics, such as LOC, McCabe, Fan-in and Fan-out, and Depth Inheritance Tree (DIT). Other researchers, like Dilek Basci and Sanjay Misra [7], designed metrics to measure the complexity of an XML schema structure.

The present research focuses on defining the quality of the XML Schemas designed for Web applications by extending the McCabe complexity metric (MCC). The old metric [6] focuses on the number of decisions regardless of the complexity of these decisions. An XML Schema has many complex features that noticeably increase the complexity of XML documents. In our new metric, we concentrate on the features that add more complexity for XML documents and that are specifically designed for Web applications. By measuring the complexity and keeping

it low, the ease of use and the maintainability of the Web applications may be improved.

When discussing the effectiveness of software, it is often discussed in the context of principles about the complexity of achieving increased functionality. We suggest certain changes to the XML McCabe Complexity Metric that are geared toward use with web-based software that implements XML documents. We propose a weighted complexity measure that gauges the level of complexity based on certain decision schemas. Thus, the aim of our metric is to add greater complexity values to schemas that are highly interdependent or cyclic. This proposed metric is important because of the widespread use of XML on the web. Also, the original XML MCC measure did not include weights for complexities that spanned successive iterations of variables in a system. Thus, it was not sufficiently sensitive for data models manipulated in XML.

This paper is organized as follows: first, major concepts about XML Schema are briefly discussed. For readers who are not familiar with XML, we recommend looking at [5]. Next, the implementation of XML McCabe Complexity metric is explained. Then, the importance of this study is discussed with a justification for these changes. At the end, a prediction of the results is proffered followed by the theoretical validation for the proposed metric.

2 XML Schema

2.1 XML Structure

Many schema languages are used to describe the structure of an XML document, such as Document Type Definitions (DTDs), XML Schema Definition (XSD), or RELAX NG. These schemas' languages are expressed in terms of constraints on the structure. In this research, we chose to study XSD because it is richer and more powerful than DTD in describing XML languages. XSDs have a rich data typing system and allow for more detailed constraints on XML documents. For example, XSDs can describe structural relationships and data types that cannot be expressed using DTDs. XSDs support namespaces and are extensible for future additions. These features make XSD a widely used language for Web applications.

XSDs should be properly designed to improve the ease of use and the maintainability of the Web applications by keeping their complexity low. Thus, it is useful and important to find ways to measure the complexity of XSD.

3 New Extended Cyclomatic Complexity Metric for XML Schema

3.1 Cyclomatic Complexity for XML Schema

In this research, we will focus on cyclomatic complexity, which was developed by Thomas J. McCabe, Sr. in 1976. By assigning numerical values to the complexity using MCC, developers may be able to evaluate the ease of use, maintainability, and quality of the XSD. This idea was proposed and developed by Lammel, Kitsis, and Remy [6]. In this paper, we extend their work by revising and improving their metric.

McCabe proposed that the complexity of a program can be determined by the complexity of the control graph, and he measured this complexity by computing the number independent control paths in the program [11]. In practice, the cyclomatic complexity (CC) of a graph (G) may be computed according to the following formula:

$$CC(G) = \#(edges) - \#(nodes) + 1$$

For a program with control flow graph G, CC(G) is also equal to the number of binary decisions in the program plus one. In [6], this method for calculating CC(G) motivates the use of MCC to calculate the complexity of XLM Schemas, and we build on [6] to develop a revised metric for calculating the complexity of XLM Schemas.

In [6], Lammel, Kitsis, and Remy explain how to apply MCC to XML Schemas, using an existing work to adopt MCC for context-free grammars. XML Schemas may be considered grammars, which means that MCC can be applied to them, and this is done by counting “decisions”. That is, the number of decisions in an XML schema is used to calculate the MCC of the schema. Lammel, Kitsis, and Remy postulated that the decision nodes in XSD include choices, element references to substitution groups, type references to types that are extended or restricted, occurrence constraints, the multiplicity of root element declarations, and nillable elements. These decisions are explained in [6] and we give the following summaries.

3.1.1 Choices

The XSD choice element is one of the model groups. It allows only one of the child elements in the <choice> declaration to be present within the containing element. The choice-model group with n branches contributes n to a schema’s MCC.

3.1.2 Element References to Substitution Groups

If h is the name of the head of a substitution group with n > 0 non-abstract participants, potentially counting h itself (if it is non-abstract), then each

element reference to h contributes n to a schema's MCC .

3.1.3 Type Reference

If t is the name of a user-defined (as opposed to primitive) type with n > 0 non-abstract subtypes (including derived types and t itself if it is non-abstract), then each type reference to t contributes n to a schema's MCC.

3.1.4 Occurrence Constraints

In MCC, if the minOccurs and maxOccurs attributes for a particle schema component have distinct values, then the component attribute contributes 1 to a schema's MCC. However,if they are the same then this component attribute does not add complexity to the schema. The default value for each one is 1 but if they have different values they will contribute 1 to the schema's MCC.

3.1.5 The Multiplicity of Root Element Declarations

Root elements are declared in the context of the schema, not within the context of a particular type. The <schema> element is the root element of every XML Schema. The number of root-element declarations is added to a schema's MCC.

3.1.6 Nillable Elements

Each nillable attribute with value "true" contributes 1 to a schema's MCC.
These decisions are illustrated in Figure 3:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- McCabe complexity is 11. -->
<xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
>
<!-- Two root-element declarations -->
<!-- The first one references a base type. -->
  <xs:element name="foo" type="type1"/>
<!-- The second one references to substitution or
derived type. -->
  <xs:element name="bar" substitutionGroup="foo"
type="type2"/>
  <xs:complexType name="type1"/>
  <xs:complexType name="type2">
    <xs:complexContent>
      <xs:extension base="type1">
      <!-- A choice with three branches -->
      <xs:choice>
      <!-- The following element is nillable. -->
      <xs:element name="branch1" type="xs:int"
nillable="true"/>
      <!-- The following element carries an
occurrence constraint. -->
      <xs:element name="branch2" type="xs:int"
maxOccurs="unbounded"/>
      <!-- There is a reference to a substitution group.
-->
      <xs:element ref="foo"/>
     </xs:choice>
    </xs:extension>
  </xs:complexContent>
  </xs:complexType>
</xs:schema>
```

Figure 3: MCC for XSD [6]

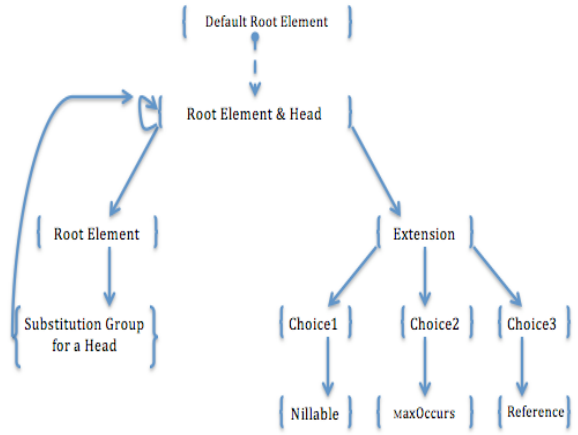


Figure 4: Tree representation for the previous schema

Lammel et al used a simple method, which is obtaining a single ordinal number totaling all decisions for each schema to determine the structural complexity of a schema as previously described. Thus, the MCC for XSD captures the "subtype polymorphism" induced complexity, which increases with the number of "subtypes". The tree representation of the schema in Figure 3 shows that each edge in the tree considered as a decision that contributes 1 to the complexity number of the schema except the dotted edge which represents the default root element for the schema.

4 Proposed Modifications for MCC

MCC measures a schema's complexity by counting the number of decisions in the schema but that does not provide sufficient information about complexity values of these decisions. The data complexity of each independent element is also important, and this is missing in MCC for XSD language. Our new and important assumption or observation is that using basic concepts and features makes the schema easier to understand, i.e., less complex. For instance, using an attribute group can improve the readability of schemas and assist in the

process of updating them because an attribute group can be defined and edited in one place and referenced in multiple definitions and declarations. Other attributes need to be redefined in case of any modification of the reference, such as derivation by restriction, which is more difficult for the schema designer. The more advanced the features are, the more complex the resulting schema will be. For this reason, in this part of the research, we will discuss the most important concepts that contribute to the complexity of XML Schemas. These features are explained below in detail.

4.1 XML Schema xsd:any Element and xs:anyAttribute

The xsd:any element and xs:anyAttribute provide the ability to extend the XML document with elements or attributes that are not specified by the schema. These features turn the XML Schema into an open content schema. With the xsd:any element, we have complete control over how much extensibility we want to allow and where. For example, suppose that we want to enable there to be at most two new elements at the top of Store's content model (see Figure 5).

```
<xsd:complexType name="StoreType">
  <xsd:choice>
      <xsd:any namespace="##any" minOccurs="0"
maxOccurs="2"/>
      <xsd:element name="Item" type="xsd:string"/>
      <xsd:element name="Seller"
type="xsd:string"/>
      <xsd:element name="Quantity"
type="xsd:gYear"/>
      <xsd:element name="RetailPrice"
type="xsd:string"/>
  </xsd:choice>
</xsd:complexType>
```

Figure 5: Using xsd:any element as a child in the xsd:choice element

In Figure 5, the xsd:any element has been placed at the top of the xsd:choice element, and it is set to maxOccurs="2". In instance documents, the <Store> content will always end with <Item>, <Seller>, <Quantity>, and <RetailPrice>. Prior to that, two well-formed XML elements may occur.

Using xsd:any element is essential for many designers because it provides more flexibility. It turns instance documents from static structures into dynamic data objects. In practical use, this element increases the complexity of dynamic access to XML documents.

The xsd:any element lets one create complexType definitions in the schema without defining the exact structure of certain parts of that complex type. When the XML Schema exists as a part of a Web service, language binding for the mapping of xsd:any element will be used. This process may not be easily done with those elements that increase the complexity. The document will be more difficult to understand and to maintain. For instance, assume that XML elements that occur in a Simple Object Access Protocol (SOAP) message should be mapped into Java objects. There are rules to determine how to map simple and complex types. If there is an xsd:any element in the XML Schema, the programmer has to be concerned with type mapping and serialization and de-serialization of data. That is, the programmer will have to know how to parse an XML element and turn it into the appropriate Java object, and vice versa.

As a result of the complexity that may be caused by xsd:any element, we will add weight to the xsd:choice decision if it has xsd:any element. Table 1 shows different tools used for mapping and binding XML documents from XML Schema into object oriented classes or relational tables to be used in data exchange.

Table 1: Shows the supporting levels of xsd:any element in different languages

Tool	Supporting Level
.NET Framework	Partially supported
Java WSDP	Recently supported with additional code
Siebel Support Level	Partially supported

4.2 XML Schema xs:redefine Element

The redefine element redefines simple and complex types, groups, and attribute groups from an external schema.

```
<xs:redefine schemaLocation="dcterms-elem.xsd">
  <xs:simpleType name="AgentScheme">
    <xs:restriction base="AgentScheme">
      <xs:enumeration value="overheid:City"/>
      <xs:enumeration value="overheid:Province"/>
      ...
    </xs:restriction>
  </xs:simpleType>
</xs:redefine>
```

Figure 6: Example of using xs:redefine element as a root element

The XSD language provides an element for updating a type definition in a process whereby the type is effectively derived from itself. The xs:redefine that is used for redefining performs two tasks. The first is that it acts as an xs:include element by bringing in declarations and definitions from another schema

documents and making them available as part of the current target namespace. The declarations and types in the schema must be from a schema with the same target namespace or one that has no namespace. Secondly, types can be redefined similar to type derivation, with the new definition replacing the old one.

By using xs:redefine, schema instances can include group or type definitions from schema documents and then pervasively change these definitions. Redefinition affects type or group definitions in the including schema and those in the included schema as well, which is why it is considered pervasive. All references to the original type or group in both schemas refer to the redefined type, while the original definition is overshadowed. This leads to a problem with the schema because redefined types can unfavorably interact with derived types and cause conflicts. A common conflict is when a derived type uses an extension to add an element or attribute to a type's content model; a redefinition also adds a similarly named element or attribute to the content model. Redefining elements can produce unexpected results, such as ill-formed definitions, on other type definitions that are based on the redefined definitions.

The .NET Framework and Java Web Services Developer Pack (Java WSDP) do not provide binding support for the xs:redefine element. Application developers have to manually create these artifacts.

For these reasons, an element with xs:redefine is assigned extra weight for that element because of the high level of complexity caused by its features.

Table 2: Shows the supporting levels of xs:redefine in different languages.

Tool	Supporting Level
.NET Framework	Ignored
Java WSDP	Ignored
Siebel Support Level	Ignored

4.3 Derivation By Restriction in Complex Types

Restriction of complex types means creating a derived complex type whose content model is a subset of the base type. Derivation by restriction in complex types is considered to be the most difficult derivation to understand. It does not map to the concepts in object oriented programming (OOP) or relational database theory, which are primary consumers and sources of XML data.

One issue in using this derivation arises from the way restrictions are declared. When deriving a complex type from another complex type by restriction, its content model must be duplicated and refined. This duplication can replicate definitions, possibly down a long derivation chain, so any modification to a derived type must be manually propagated down the derivation tree.

As previously mentioned, using derivation by restriction in complex types is incompatible with the notion of derivation in an object oriented sense. Certain aspects of derivation by restriction do not map well to tables in a relational database, and XML Schemas that use derivation by restriction are more difficult to model as classes in an OOP language. This causes mismatch, which occurs when trying to map the contents of an XML document into a relational database or convert an XML document into an instance of an OOP class.

This feature is one of the most complex features in the XML Schema. It often causes bugs during the mapping process. It can be seen why this feature should warrant additional weight when computing the complexity of an XML Schema.

Table 3: Shows the supporting levels of complex type derivation by restriction in different languages.

Tool	Supporting Level
.NET Framework	Partially supported
Java WSDP	Partially supported with bugs
Siebel Support Level	Partially supported with bugs

The result of the previous discussion is that counting the schema decisions but neglecting the internal architecture of the decisions does not adequately capture the complexity of a given schema. In other words, two schemas having the same number of decisions may have much different complexities since the internal structures may be different. These difference complexities can, however, be measured by using the new metric which is defined in the next section.

5 Extended McCabe Complexity Metric

5.1 Implementing the Modifications

We have explained how Lammel, Kitsis, and Remy apply the McCabe complexity metric to an XML Schema when all decisions are assumed to be equally complex. However, it is not the case that all decisions are equally complex. We examine complex decisions, and we assign weights to types of decisions based on their complexity. In this manner we arrive at a new and improved metric which is an extension of the McCabe Complexity Metric for XML Schemas.

The complexity caused by using these features increases the complexity of the programs that are written to manipulate the data in the schema. If one of

these features appears in one of the previous decisions, we will assign it a specific weight. By adding weights to these decisions, we make the MCC number more reliable in measuring the ease of use and the maintainability of XML Schemas.

The complexity metric of Lammel et al is expressed as:

$$MCC(XSD) = |D| \quad (1)$$

where D= the decisions

As previously explained, the complex features of the XML Schema can affect the schema and add complexity for the schema at different levels. Based on this study, D1, D2, and D3 are decision groups in that the decision types which will be in D1 are considered less complex than those in D2, and those in D2 are considered less complex than those in D3 as described below:

- D_1, decisions
- D_2, if decision includes xsd:any element or xs:anyAttribute
- D_3, if one of the schema elements is declared by using xs:redefine or derived by restriction

We can calculate the complexity of the XML Schema using the extended McCabe cyclomatic complexity metric using the following function:

$$EMCC(XSD) = |D_1| + 2|D_2| + 3|D_3| \quad (2)$$

The proposed metric measures the complexity using the number of the decisions with a specific weight for each decision. The values of these weights depend on the complexity level of each complex feature. Thus, the xsd:any element and xs:anyAttribute will be assigned a weight of 2. The reason is the complexity of these features is not related to any other element in the schema since there is no reference or derivation related to this feature. Thus, this feature will affect the attribute itself but not any other attribute in the schema. On the other hand, the other features (xs:redefine and derivation by restriction in complex types) will be assigned a weight of 3. The complexity added by these two features affects other components in the schema. As explained earlier, the xs:redefine element modifies another components of another schema and restriction by derivation in complex types affects other complex types and increase the inheritance depth.

For the tree representation for the schema, for Lammel et al we will have a weight of one of each edge. However, for EMCC these weights are modified based on the features' complexity. In the new resulting tree, where every edge has a weight of 1, 2 or 3, we simply add all that weights up. These weights help to determine the actual complexity of a given schema according to the number of decisions and the structure of the schema. That makes the resulting complexity number more reliable by specifying the complexity of the structure. Nevertheless, there is a good justification, as given in the paper, for the proposed weights, but it is also clear that future research may justify modifying the weights.

6 Validating the Proposed Metric

The theoretical validation of a proposed metric helps justify the use and value of complexity metrics. We use the description of properties of Briand et al [8]. In this section we will validate if our proposed metric satisfies the five properties of a complexity metric.

The elements of E are the default root element for an XML schema, XML schema decisions, and leaves. More exactly, the leaves are the possible final or non-choice outcomes from the decisions. For example, if we have the leaf node "nillable," this leaf will be instantiated with a "true" or "false."

The relationships or arcs, which are the elements of R, represent choices which need to be made. Thus, arcs only come from choice nodes. There may be choice nodes with only one arc coming from them. For example, if the choice is nillable, then only one arc will come from this node. It should be noted that by convention no edge comes from the default root element.

Given a system S = (E,R) ,where E= elements and R= relationships (edges), the MCC for an XML schema is the number of edges in S.

We want to verify the five properties for complexity metrics given in [8].

Property Complexity 1 Non-negativity: The complexity of a system S = <E, R> is non-negative if:

$$EMCC(S) \geq 0$$

Since our metric counts the number of edges in S, then our metric always returns 0 or a positive value.

Property Complexity 2 Null Value: The complexity of a system S = <E, R> is null if

R is empty.

$$R = \varnothing \Rightarrow EMCC(S) = 0$$

If R is empty, then our metric returns the value 0.

Property Complexity 3 Symmetry: The complexity of a system S = <E, R> does not depend on the convention chosen to represent the relationships between its elements:

$$\left(S = \langle E,R\rangle \; and \; S^{-1} = \left\langle E,R^{-1}\right\rangle\right) \Rightarrow EMCC(S) = EMCC(S^{-1})$$

Since R and R^{-1} have the same number of elements, then our metric returns the same answer for S and S^{-1}.

Since Properties 4 and 5 involve modules, we define a module of S to be any subgraph of S. Let T be a subgraph of S; let ET be the elements in T; and let R_T be the relationships in T. Then EMCC(T) is the number of relationships in R_T, i.e., EMCC(T)= $|R_T|$.

Property Complexity 4 Module Monotonicity: The complexity of a system S = <E, R> is no less than the sum of the complexities of any two of its modules with no relationships in common:

$$(S = \langle E, R \rangle \text{ and } m1 = \langle E_{m1}, R_{m1} \rangle$$
$$\text{and } m2 = \langle E_{m2}, R_{m2} \rangle$$
$$\text{and } m1 \cup m2 \subseteq S \text{ and } R_{m1} \cap R_{m2} = \varnothing)$$
$$\Rightarrow EMCC(S) \geq EMCC(\mathrm{m1}) + \mathrm{EMCC}(m2)$$

Let U and V be two modules in S such that R_U and R_V have no relationships in common. Since every relation that is in R_U or in R_V is also in R and since no relationship is in both R_U and R_V, then clearly EMCC(U) + EMCC(V) ≤ EMCC(S). In fact, EMCC(S) may be much larger than EMCC(U) + EMCC(V) because R may contain many relationships which are in neither R_U nor R_V.

Property Complexity 5 Disjoint Module Additively: The complexity of a system S = <E, R> composed of two disjoint modules m1, m2, is equal to the sum of the complexities of the two modules:

$$(\mathrm{S} = \langle\ \mathrm{E}, \mathrm{R}\ \rangle \text{ and } \mathrm{S} = \mathrm{m1} \cup \mathrm{m2} \text{ and } \mathrm{m1} \cap \mathrm{m2} = \varnothing)$$
$$\Rightarrow \mathrm{EMCC}(S) = \mathrm{EMCC}(\mathrm{m1}) + \mathrm{EMCC}(\mathrm{m2})$$

Let U and V be two modules in S such that R_U and R_V have no relationships in common. Since every relation that is in R_U or in R_V is also in R and since no relationship is in both R_U and R_V, then clearly EMCC(U) + EMCC(V) ≤ EMCC(S). In fact, MCC(S) may be much larger than EMCC(U)+EMCC(V) because R may contain many relationships which are in neither R_U nor R_V.

7 Summary

Extensible Markup Language (XML) is a universal format for data and has been adopted for exchanging information among distributed applications. This research proposes a new complexity metric for XML Schema documents (XSD) based on the McCabe complexity metric (MCC), which is used to measure cyclomatic complexity and on the work of Lammel, Kitsis, and Remy, who showed how the McCabe metric could be applied to XML schemas. Many schema languages are used to describe the structure of an XML document. XSD was chosen for the study because it is richer and more powerful than others.

The XML Schema is the structural representation of an XML document. MCC is used to measure the complexity of ordinary programs and needs to be adapted to XSD. Our new metric measures the complexity using the number of the decisions with a specific weight for each decision. Thus, the xsd:any element or xs:anyAttribute is assigned a weight of 2, and the other features (xs:redefine and derivation by restriction in complex types) are assigned a weight of 3.

These weights help to determine the actual complexity of a given schema according to the number of decisions and the structure of the schema. This means that the resulting complexity number is more sensitive to the complexity of different parts of the structure. The proposed metric is validated based on five complexity properties.

8 Acknowledgement

I am extremely grateful to Prof. Austin Melton for his support and suggestions throughout the research and writing for this paper.

9 References

[1] http://msdn.microsoft.com/en-us/library/ms256235%28v=vs.85%29.aspx
[2] http://www.xml.com/pub/a/2003/10/29/derivation.html
[3] http://msdn.microsoft.com/en-us/library/ms256142.aspx
[4] http://www.xfront.com/ExtensibleContentModels.html#BestPractice
[5] http://www.w3.org/
[6] R. Lammel, S. Kitsis, and D. Remy, "Analysis of XML schema usage", Proceedings of XML, 2005.
[7] D. Basci and S. Misra, "Complexity metric for XML schema document", Proceedings of the 5th International Workshop on SOA and Web Practices, 2007.
[8] L. C. Briand, S. Morasca, and V. R. Basily, "Property based software engineering measurement, IEEE Transactions on Software Engineering, Vol. 22, 1996.
[9] E. van der Vlist, XML Schema, O'Reilly Publication, Sebastopol, 2002.
[10] T.J. McCabe and C.W. Butler. Design complexity measurement and testing. Communications of the ACM, 32:1415–1425, 1989.

An Improved Model of Configuration Complexity

Moheeb Alwarsh
Department of Computer Science, Kent State University, Kent, OHIO, USA
malwarsh@kent.edu

Abstract - *Studies have been conducted on system configurations to find metrics to calculate configuration complexity. These studies help in quantifying the complexity, estimating the required manpower and the related cost for implementing these configurations. However, a problem arises when these metrics are not precise enough to cover all aspects of system configuration. This paper is a continuation of work done by others [1][2]; we develop more precise metrics for calculating configuration complexity.*

Keywords: Configuration Complexity, Complexity Metrics

1 Introduction

Companies pay tremendous attention to data centers due to their strategic importance in the success of the companies. Operational costs for system configuration and installation in these centers is increasing day after day, and in some cases, these costs exceed those of the hardware and software [5]. Researchers are working on ways to decrease the operational costs or at least, to stabilize them from further future increases. Studies have been implemented in this area to reduce operational activities with self-healing benchmark or to develop a model for configuration complexity like [7] and [2], but the problem of determining the human costs when planning for new development is still difficult to implement without better tools and methodologies.

To understand the complexity of estimating the operational costs of configuration, we first have to know the configuration and maintenance processes used when developing a system. A system is composed of several components connected together to form an interface for a particular service. As an example, to install and configure a proxy server to provide Internet service, we need to install and configure one or more operating systems like Linux, a firewall, a Dynamic Name System (DNS), a Network Time Protocol (NTP), the proxy application, Lightweight Directory Access Protocol (LDAP) for accountability, and a load balance application or appliance to distribute user loads on the system. The human cost of implementing such a system depends on how many persons are needed and for how long. Of course, configuration complexity varies from system to system based on the time of installation and configuration. Some systems can be installed in few steps and a short time, but others need a tremendous effort to be implemented. An expert administrator can finish the job faster than a novice administrator. There is a need to develop and implement an algorithm that measures configuration complexity and that fits any expertise of manpower.

This paper is a continuation of work done by others [1][2] to quantify configuration complexity. In addition, others have proposed a mechanism to translate the result of quantification to a measure used to estimate the cost of manpower. This estimation is not precise and it could produce unexpected results. We propose improvements in the way the configuration complexity is quantified to provide a more precise mechanism.

The remaining parts of this paper are organized as follows. Section two presents the Complexity Quantification method used in [2]. Section three discusses the improved model of quantifying Configuration Complexity. The fourth Section is Related Work and the last will include the summary and concluding remarks.

2 Related Work

Ad hoc configuration is the main cause of many configuration issues, and ad hoc configuration also makes the fixing of problems discovered late more complex. This affects the effort in completing the configuration process and increases the implementation time which subsequently increases the cost [3]. Incorrect configurations cause around 90% of these problems [6]. Therefore, planning a configuration document that includes all entered parameters is necessary to help avoid these costs.

In [1], three distinct types in the configuration process lifecycle are identified. The first is an initial configuration where performance is not considered until the end of the process. The second configuration type is when the performance of a running system decreases, and in this case a configuration is needed to put it back on the track. The last type of configuration is implemented when a new performance level is required. However, an implemented configuration without a validation check would increase the complexity of debugging the system. Validation needs to be included in the configuration process lifecycle.

Configuration complexity metrics have been introduced in [2], and they are classified into three areas. They are execution complexity, parameter complexity and finally memory complexity which is human memory. [2] introduces an approach to measure the complexity; high complexity

increases the probability of getting a defected system. Implementing a configuration which depends on measuring the system performance at the end of the configuration process [1] might increase the complexity of finding and fixing errors when more debugging time is required to fix problems, and this time may exceed the time needed to re-configure the entire system. Others [4] have built their cost prediction on these metrics, and this could provide a less precise estimation because problems that might be encountered after the configuration process are not considered in calculating the costs. In addition, increased memory complexity could mean an increase in human error during the configuration process by a human operator.

We introduce an improved model that uses two configuration complexity metrics from the [2] and one new category for validating the configuration. Our model provides more precise metrics to quantify the configuration complexity.

3 Complexity Quantification

In [2], the configuration complexity measure is based on three components. These components are collected from a manual configuration of an e-commerce solution; see Figure 1. The first component is execution complexity, and it consists of two metrics. The first metric is the number of actions for the configuration procedures. Borrowing an example from [2], to configure an e-commerce system (see Figure 1), we need to perform 59 human actions. The second metric is the context switch metric which increments when a user temporarily stops configuring one component and switches to configure another component.

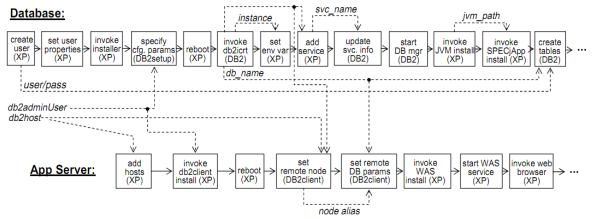


Figure 1. Partial Manual Configuration and Action Sequence for Installing e-Commerce System [2]

The second component of configuration complexity is the parameter complexity. This component consists of five metrics.

Parameters Count: This is the total number of parameters involved in the installation and configuration procedure.

Parameters Use Count: This is the total number of times the parameters are used in the procedures. For example if we use one parameter in three locations, then our Parameters Use Count value is 3.

Parameters Cross Context: If we use a parameter in configuring one component and use the same parameter in configuring another component, then we have a total of 2 Parameters Cross Contexts.

Parameters Adapting Count: This is the total number of parameters used in one form then adapted to be used in other forms. An example is the fully-qualified path name where the path name changes based on source location. Assume we have a directory /a/b/c/d as the target directory. Then if a source resides in "a" directory, the path will be "/b/c/d". If a source resides in "c" then the path is "/d". The Parameters Adapting Count in this case is 2.

Parameter Source Score: Each parameter is assigned a score from 0 to 6 based on how difficult it is to obtain the parameter. For example, a parameter that would be obtained as a result of executing several commands is more difficult than a parameter that could be entered directly like user name.

The third component is memory complexity, and it refers to the number of parameters a human operator must memorize or remember during configuration. The memory complexity is based on three metrics: Memory Size, Memory Depth and Memory Latency with the value of each being an average. The configuration approach in [2] assumes that a system administrator already knows how to configure an e-commerce solution. Memory Size means the remaining parameters which a system administrator needs to memorize and is required to use for each step in the configuration. For example, Figure 1 shows the process of configuration. At the first step a system administrator will memorize the created user and profile location. These parameters will be used later in the configuration. In this case the memory size is 2. Storing parameters is based on Last-In-First-Out LIFO with non-associative lockup. Memory size, which is the size of the stack that has all memorized parameters needed for current or future configuration, is captured prior to each configuration action. The Memory Depth is the process of measuring the depth in the stack for a targeted parameter. For example, if we have a stack of 10 parameters and the order of the required parameter is 5; then, memory depth at this stage is 5. Memory Latency is calculated based on the time between storing a parameter and using it. For example, if we use "user name" in Figure 1 at the end of the installation, then the Memory Latency will be the total time between storing the user name and using it. Since values are fluctuating up and down, only the maximum and average of each metric will be calculated. Figure 2 shows the metrics collected from the three memory complexity components after configuring an e-commerce solution; Figure 1.

	Measure	Value (manual procedure)
Exec	NumActions	59
	ContextSwitchSum	40
Param.	ParamCount	32
	ParamUseCount	61
	ParamCrossContext	18
	ParamAdaptCount	4
	ParamSourceScore	125
Memory	MemSizeMax	8
	MemSizeAvg	4.4
	MemDepthMax	12
	MemDepthAvg	1.5
	MemLatMax	55
	MemLatAvg	4.4

Figure 2. Configuration Complexity Metrics measures [2]

4 Improved Model for the Configuration Complexity

In section 2 we have demonstrated how to calculate the configuration complexity metrics using the procedure proposed in [2]. However, there are some concerns with the proposed method when it comes to memory complexity and the lack of a validation plan for configuration. We will discuss memory complexity proposed by [2]. Then, we will introduce our proposed validation metric and show how it could help in reducing the debugging time and the configuration complexity process.

4.1 Memory Complexity

Memory complexity is not applicable in all scenarios when configuring a new or an existing system. For example, after planning and designing a phase of a new system, a system administrator should write down all successful steps required to install and configure the new system. Reproducing the same steps is necessary for automation, validation and quality assurance. Configuring a complex system might require several hours or days in collaboration with other system administrators. Depending on a human operator's memory to install and configure a new system is very risky and might endanger the entire process of installation and configuration. An example is when a wrong parameter is selected or entered without discovering it until the end of the configuration process. Memory complexity is not a precise metric that should be used in all configuration scenarios. Using memory complexity with large systems would increase the probability of human errors. There is a need to document the necessary steps for configuring a new system or reconfiguring an existing system to facilitate revision of the implementation process, automation and validation.

4.2 Validation Complexity

A complex system might consist of components that have a high degree of context switching during configuration similar to Figure 1 or a low degree of context switching similar to Figure 3. However, the complexity of validating the functionality of each component increases after building the entire system. If no testing for validation of functionality is implemented right after completing the configuration of each component for a complex system whenever this is possible, then the time of debugging and verifying could increase significantly; in fact, it might exceed the time of configuration. An example is when debugging errors of a node in a Rocks cluster. Reinstalling the node could be implemented in one or two steps but debugging the errors might take more steps. This would increase the operational time and, therefore, would likely increase the cost.

Let V be a set of validation procedures that are not part of system components where V = {V1, V2, V3...Vn}; let SC be a system's components where V ϵ SC; and let S be a system where S = {SC1, SC2, SC3...SCm}. Then, when the system is complete, the potential complexity of validating the functionality of the entire system is much higher if no incremental validation of each system component has been implemented. For example, let us assume that we have a procedure of three steps to validate the functionality of a system component and we have five system components. If we build the system without putting any stop points to check what has been implemented, then we might end up with a defected system that needs to be debugged in order to find the cause of the error.

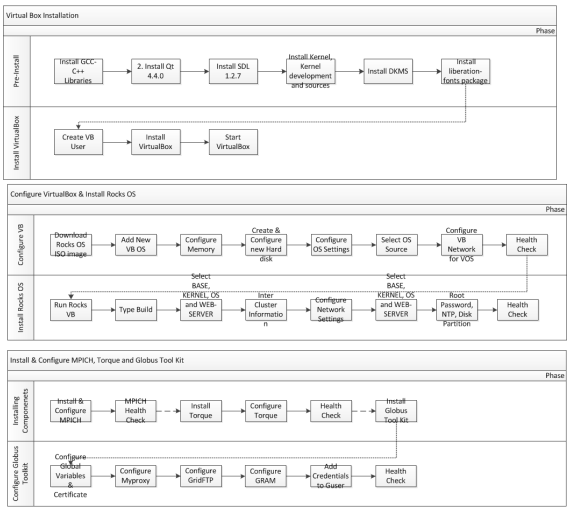


Figure 3. A semi sequential process for installing Grid on top of a virtual cluster.

We assume each system component is implemented as a black box that works independently. A system administrator combines some system components to provide a working solution. For example, a company may want to regulate Internet access for its employees and at the same time provide a layer of protection. To implement this solution, the company would need a proxy application that works as software under an operating system or in an appliance as an independent solution. The proxy needs a Domain Name System (DNS), firewall, a Network Time Protocol (NTP) and a Lightweight Directory Access Protocol (LDAP) or any other user accountability solution. Once we configure all these system components and start the system, it might work and it might not. For the latter case, we have to check each system component and make sure it is working well. For example, we can start with the DNS by issuing this command (nslookup www.domain.com). If the DNS is not working, then maybe the firewall is causing the issue or the proxy application corrupted some DNS files during installation or other system component is causing the problem. Sometimes one component might cause the problem, and in other cases the integration of two or more components might cause the problem. The system components developers didn't sit together to provide a single working solution. Instead, each one focused on

providing an independent solution that could be used with other applications. In the proxy example, if we assume that the operating system is functioning without any problem, then we have five system components that need to be tested if the system fails. If we assume that each system component needs three steps to verify its functionality, then the total number of validation checks is 15 for the best case and 210 for the worst case.

If we just check the functionality of each system component, then each one will take 3 validation checks with a total of 15. There might be only one system component that causes all the issues. For example we might remove the firewall and test the system functionality. In this case we will end up with validating the remaining four system components (DNS, NTP, LDAP and the proxy Application) to make sure they are working well. This scenario might be applicable to other components, and this could produce 16 system component validations with a total of 48 validation checks. This could lead to do a total of 210 validation checks in the worst case by disassembling all system components. As an example, let's assume that there is only one system component out of 5 which is causing a problem and affecting other components. We will try to remove one by one and check all other components to make sure that a certain component is causing the problem. If we start by removing system component one {1}, then we have to test the remaining 4 components {2,3,4,5} to make sure they are free from error. If we found that the problem is still there, then we will put back {1} and remove {2}, then we will check the remaining system components {1,3,4,5}. If no is problem found we will keep this process until finding the one that is causing the problem. If system component {5} is the one that is causing the problem then it will cost 48 validation checks. What if we don't know if a combination of more than one system component is causing the problem? Then, we will end up testing all possibilities and wasting a lot of time. The calculations below show the possibility of finding the problem. First, we test the system as whole, then we might remove system components one by one or a combination of more than one.

{1, 2, 3, 4, 5} : 5 * 3 = 15

{1,2,3,4}, {1,2,3,5},{1,3,4,5}, {2,3,4,5} : 16 * 3 = 48

{1,2,3}, {1,2,4},{1,2,5},{1,3,4}, {1,4,5}, {2,3,4}, {2,3,5}, {3,4,5} : 24 * 3 = 72

{1,2}, {1,3}, {1,4}, {1,5}, {2,3}, {2,4}, {2,5}, {3,4}, {3,5}, {4, 5} : 10 * 6 = 60

{1}, {2}, {3}, {4}, {5} : 5 * 3 = 15

Total = 210 validation check

A representation of this calculation is shown on Figure 4. The worst case occurs when we encounter an error after the end of the configuration process and we start by testing system components one by one, then combining system components to figure out the combination of system components that makes this problem.

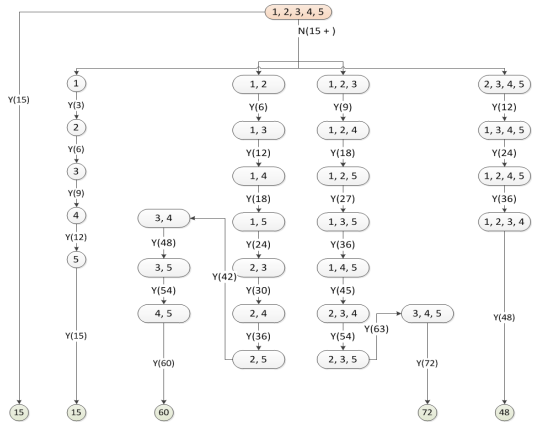


Figure 4. Probability that one or more system components are causing a problem. Worst case is when the problem occurs by combining system component {1,2, 3, 4} and we start from left of the tree to the right. In This case the worst case is 210 validation check.

We can't measure the cohesiveness in this scenario because we don't know how a system component is designed and what might affect this design. To reduce the total number of validation checks, we need to implement a stop point after assembling a system component to make sure that it is working well and doesn't cause any problems by itself and with the previously assembled components. If errors are found with a newly configured system component, then a backward validation starts building from the last SC toward the first one to find if there is any conflict. By following this procedure, we will reduce the number of validation check to the minimum possible validation check. Let's use the previous example that we illustrated early regarding finding a defective system component out of 5. We will start by configuring SC {1} and implement a validation check. If no problem found, then we will move forward to configure SC {2} and implement validation check. We continue to add SC {3}, {4} and then {5} where we discover the problem. In this case the total validation is 15 compared to 48 with the previous example. If for some reason, we found an error during the configuration process, then we implement a backward validation check between the newly added SC and the previous SC. In this case we would get 15 as a best case and 54 as a worst case compared with 210. See Figure 5.

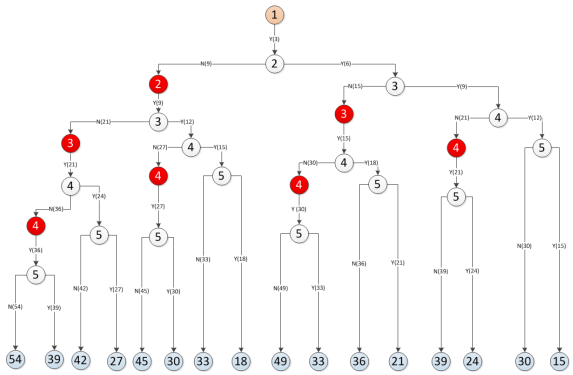


Figure 5. Probability that one or more system components are causing a problem. The minimum is 15 validation checks if there is no error. The maximum is 54, and this occurs when an error is discovered in each component which requires backward validating check.

Validation complexity can be measured with two metrics. The first one is the number of validation checks needed for each system component. The second metric is the total number of validation checks for the best case and worst case. By combining the validation complexity metrics with the execution and parameters metrics (See Table 1), we would get a more precise quantitative measure of the configuration complexity because we would be able to get more details about the problems that we might encounter during the configuration processes. Assuming the process of configuration is implemented without errors is not valid [6]. This would allow us to estimate a more accurate time for the configuration process and therefore its cost. Memory complexity is not a realistic metric because a configuration process for a complex system is documented for verifications, quality assurance, and to reproduce the configuration steps. Thus, the memory complexity is not normally applicable. Further, consider, for example, when configuring a complex system that has hundreds or even thousands of parameters, then documenting all processes of configuration in a sequential steps would reduce human errors, provide a mechanism for other administrators to review the process and configuration process can be shared and then implemented by more than one system administrator. For example, let us assume the following which admittedly is extreme. We want to improve the performance of the US aviation system and we have only 10 minutes to implement the new configuration. We have 2 minutes for entering 200 parameters and 3 minutes for restarting the system. If things go wrong, then we can roll back by returning all old parameters in the same sequence we replaced them and it will take 2 minutes for entering the parameters and 3 minutes for restarting. We have a window of 10 minutes to complete our reconfiguration; otherwise, a back-up copy of the system will be installed which will take more than 2 hours. During these two hours, large airplanes could not fly in the US. We can see that depending on a system administrator's memory for this task is not really an option. Also, it is too risky to waste 2 hours of no fly zone which will cost a lot of money. Memory complexity is not an option in this scenario and the entire configuration steps need to be documented from the start to the finish before the implementation.

Execution Complexity	Parameter Complexity	Validation Complexity
Number of Actions	Parameters Count	Number of validation for each component
Context Switch	Use Count	Total Number of Validation (worst & best case)
	Cross Context	
	Adapting Count	
	Parameters Score	

Table 1. Improved Model of Configuration Complexity

5 Concluding Remarks

We propose a new and improved incremental validation metric for quantifying configuration complexity. This metric is based on the number of validation checks for each component and the total number of validation checks. Also, we introduce an improved model to increase the precision of the output. We removed memory complexity metrics from [2] and add validation complexity; this improved model is based on work done by others. We have shown that a better approach, which requires a validation check for all components after each configuration process to make sure all components are working well before moving further and configure a new component, would decrease the debugging time and reduce the complexity of the configuration process. This would help in providing a more precise estimate of manpower cost. The validation metric consists of two parts. The first part is to calculate the number of validation checks for each system component. The second part is to calculate the probability of best and worst case scenarios for implementing the configuration process.

Acknowledgments

Thanks to Dr. Austin Melton for his inputs, thoughts and valuable suggestions in this paper.

6 References

[1] Aaron B. Brown, J.L. Hellerstein. "An approach to benchmarking configuration complexity". In Proc. ACM SIGOPS European Workshop 2004.

[2] Aaron B. Brown, Alexander Keller, Joseph L. Hellerstein. "A model of configuration complexity and its application to a change management system". Integrated Network Management 2005: 631-644.

[3] Afzal, Uzma. Hyder, S. I. "Configuration Complexity: A Layered based Configuration Repository Architecture for conflicts identification". GJCST 2010: 66-71.

[4] Yixin Diao, Alexander Keller, Sujay S. Parekh, Vladislav V. Marinov. "Predicting Labor Cost through IT Management Complexity Metrics". Integrated Network Management 2007: 274-283.

[5] T. Eilam, M. Kalantar, A. V. Konstantinou, G. Pacifici, J. Pershing, A. Agrawal. "Managing the configuration complexity of distributed applications in Internet data centers". IEEE Communications Magazine 2006:166-177.

[6] Kalapriya Kannan, Nanjangud C. Narendra, Lakshmish Ramaswamy. "Managing Configuration Complexity during Deployment and Maintenance of SOA Solutions". IEEE International Conference on Services Computing (SCC) 2009:152-159.

[7] Aaron B. Brown, Charlie Redlin. "Measuring the Effectiveness of Self-Healing Autonomic Systems". ICAC 2005: 328-329.

Role of Function Point as a Reuse Metric in a Software Asset Reuse Program

Johns T. Joseph[1]
[1] Department of Computer Science, Kent State University, Kent, Ohio, USA

Abstract - *The role of Function Point as a reuse metric in a software asset reuse program is analyzed in this paper. The paper takes a practical implementation as an example and walks through the issues faced by the developers and managers in using Function Point as a measure of software asset reuse. Productivity measures such as "cost saving" and "actual reuse" are generated to study the productivity of the software development firm. The author highlights the practical implementation issues, and recommends resolution options to overcome these issues. The paper also highlights the need of standardization of software reuse metrics as a step towards establishing software development as an engineering discipline*

Keywords: Software metrics, software reuse, function points, productivity, cost saving

1 Introduction

As the software development organizations are getting more competitive, there have been lots of research initiatives to identify and use efficient metrics to measure the productivity and quality of software development. Software reuse can be defined as the use of existing software artifacts to improve the productivity and quality of new software. There can be different types of software reuse. Following are some of the types of software reuse:

a) *Code Reuse* – The code snippet is reused in new software. It is easy to reuse; however, this practice does not scale well.
b) *Template Reuse* – Design and coding standard templates are reused.
c) *Library Reuse* – The library and software components are reused in a systematic fashion.
d) *Framework Reuse* – New functionality is developed over a reusable framework. Use of a framework allows a standardized way of addressing common issues as well as implementing good practices.
e) *Design Pattern Reuse* – Design patterns are published and used in software development.

There are lots of challenges in effectively implementing a Software Reuse program in an organization. The technology changes quite frequently. For example, a software asset in DotNet technology cannot be used in Java technology. The teams working on project should consciously develop and design the software in a reusable fashion. This takes additional design and development effort. Another aspect of software reuse is an efficient way of maintaining the library of software, its various versions, and how they can be used. Organizations having a better software asset management system can reuse software better than others. There are other challenges such as political and psychological aspects that come into play during software reuse.

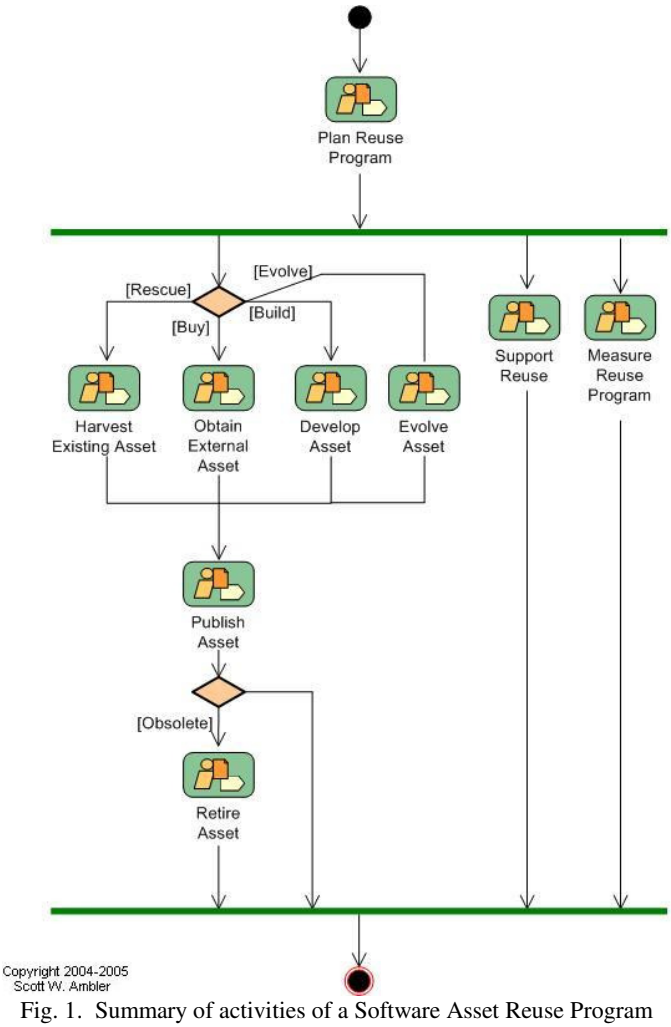


Fig. 1. Summary of activities of a Software Asset Reuse Program

Scott W. Ambler [22] depicted the software reuse in a very clear manner in Figure 1. The diagram summarizes the

reuse and management of a software asset. The following activities in software asset management are important to increase the productivity of the organization:

- Publish, categorize software assets
- Control asset versions and manage dependencies
- Locate assets
- Improve communication across teams
- Track usage & measure reuse

As we all know, to measure an effective software reuse program, we need a very well defined reuse metric. Lots of research and papers ([8], [4], [9]) have highlighted that Function Point can be used as a software productivity measure. This paper highlights the issues in using Function Point as a unit of measure for software reuse in a practical environment, and proposes how it can be enhanced to an efficient usage.

Some of the reuse metrics and models proposed by [8] are cost benefit analysis, maturity assessment, amount of reuse, failure modes analysis, reusability assessment and reuse library metrics. There are lots of ways of measuring reuse metrics; however, there is no standard accepted methodology in the industry. Everyone, however, does realize that software reuse improves the quality and productivity of the organization.

2 Software Reuse – A Practical Study

A software reuse metric using Function Points as the basis was used in an actual product development organization. This section summarizes the implementation of a software reuse program for a period of two years. The calculation of the Function Points was based on the IFUP Counting Practices Manual 4.1. One of the reasons for choosing FP as the metric was that it is a standard way of denoting the size of a software asset. It is better than LOC (Line of Code) to depict the size of the software [11]. It is easier to implement and less complex than other variants such as Feature Point, 3D Function Point, Mark II Function Point, Full Function Point, COSMIC Full Function Point as summarized in [1]. FP falls under direct measurement classification. The three different measurement classifications are described in [6]. The inter-rater reliability issue of Function Point was resolved through a practical study in [13].

A dedicated team was involved in developing reusable software assets. A software asset is certified for reuse after the component is reused in a product and passes Alpha testing. The Alpha testing was conducted by an independent team at the developer's site. The software asset is registered in an in-house software asset management system. This helps in the discovery of the reusable assets by the developers. Once an asset is discovered the dependencies and location are retrieved from the system. This information is used by the users of the software asset.

The reusable assets were developed using different technologies such as .NET based web service, ASP.NET, C#, Microsoft SQL Server, Transbase database, Oracle database, Microsoft SQL Server Integration Service (SSIS, an ETL tool), Java Swing, Applet, Java J2EE, Adobe Flash, AIR (Adobe Integrated Runtime), JSP (Java Server Pages), XML, XQuery, Oracle Data Integrator (ODI, an ETL tool). Since Function Point metric is "mostly" technology agnostic, it helped the team to measure the reuse of the varied technology based assets in a common and efficient manner.

Once a reusable asset is certified, the lead designer of the asset calculates the FP and its reuse percentage in the product. Refer [1] and appendix of [2] for details of the Function Point terminologies used below such as EI, EQ, EO, ILF, ELF, Degree of Influence. The author designed a template to calculate the FP in an efficient manner. The template has Excel based macros that enhances the ease of calculation. The process is broken into following steps:

- *Step 1*: Identify Use Cases or Functionality
- *Step 2*: Categorize Uses Cases as EI, EQ, EO, ILF, ELF
- *Step 3*: Derive the complexity of use cases and assign rating
- *Step 4*: Summarize all the uses cases in different categories and assign weight-age based on its category
- *Step 5*: Derive Degree of Influence
- *Step 6*: Generate Total Function Point of the reusable asset. Potential Reuse is the total FP of the reusable asset
- *Step 7*: Actual Reuse is calculated by identifying the percentage use case (FP) reused in a product

Use Case / Functionality	Tables FTRs	UI Elements DET	EI Rating
<Use Case No.>-<Use Case Desc / Functionality>	1	16	Avg
<Use Case No.>-<Use Case Desc / Functionality>	1	1	Low
<Use Case No.>-<Use Case Desc / Functionality>	1	2	Low
<Use Case No.>-<Use Case Desc / Functionality>	2	20	High
<Use Case No.>-<Use Case Desc / Functionality>	1	5	Low

Fig. 2. Step 2a - The uses cases or functionality is categorized as External Input

Criteria			
File Type Referenced (FTR)	Data Elements (Controls)		
	1-4	5-15	>15
< 2	Low	Low	Avg
2	Low	Avg	High
> 2	Avg	High	High

Fig. 3. Step 3a – Derive the complexity based on the table shown in the figure for EI (External Input)

Figures 2 & 3 show a portion of the template that is used to calculate the FP. In these two figures the identification of use cases as External Input and assignment of the complexity to these uses cases is shown. There are similar portions in the template for External Inquiry, External Output, Internal Logical File and External Logical File categories. Figure 4 shows the portion of the template that summarizes all the use cases /functionality under the transactional function (EI, EQ, EO) and data function functions (ILF, ELF).

Figure 5 shows the portion of the template where the degree of influence is derived by assigning a rating to each

General System Characteristic (GSC). This section calculates the technical complexity of the software asset. The 9th GSC is "complex processing". This is however very loosely defined and impacts the overall Function Point in a very minor way. The Unadjusted Function Point (UFP) is derived in step 4. The UFP is the size of the software component that depicts only the data flow complexity; it does not take account of system characteristics such as end user efficiency or installation ease.

		Transaction Functions		
Use Cases	Rating	Input Forms EI	Simple Reports EQ	Calc Reports EO
Total Use Cases	Low	3	2	2
	Avg	1	1	1
	High	1	2	2

		Data Functions	
Tables	Rating	Internal Tables ILF	External Tables ELF
Total Tables	Low	2	2
	Avg	2	2
	High	1	1

Fig. 4. Summary of the uses cases

General System Characteristics (GSCs)	Degree of Influence (DI) 0 - 5
1. Data Communications	4
2. Distributed Data Processing	0
3. Performance	4
4. Heavily Used Configuration	2
5. Transaction Rate	2
6. Online Data Entry	2
7. End-User Efficiency	5
8. Online Update	5
9. Complex Processing	1
10. Reusability	0
11. Installation Ease	2
12. Operational Ease	1
13. Multiple Sites	2
14. Facilitate Change	0

Fig. 5. Deriving the degree of influence by assigning ordinal rating to the General System Characteristics (GSCs)

Figure 6 shows the formula used to calculate the overall function point. It is mentioned below,

$$VAF = (TDI * 0.01) + 0.65 \quad (1)$$

where,

VAF = Value Added Function Point

TDI = Total Deg. of Influence based on GSC ratings

The total Function Point is then derived by following formula

$$FP = VAF * UFP \quad (2)$$

where,

FP = Function Point

VAF = Value Added Function Point

UFP = Unadjusted Function Point

Figure 7 shows that the use case percentage reused in a product was identified. The percentage usage is applied to the asset's Function Point to derive the actual reuse value.

Type of Component	Complexity of Components			
	Low	Avg	High	Total
External Inputs (EI)	*3	*4	*6	19
External Outputs (EO)	*4	*5	*7	27
External Inquiries (EQ)	*3	*4	*6	22
Internal Logical File (ILF)	*7	*10	*15	49
External Logical File (ELF)	*5	*7	*10	34
	Total Number of Unadjusted Function Points (UFP)			151
	Total Degree of Influence (TDI)			30
	Value Adjustment Factor (VAF) VAF = (TDI * 0.01)+ 0.65			0.95
	Total Adjusted Function Point FP = VAF* UFP Potential Use			143

Fig. 6. Calculate the final function point. This value is termed as "Potential Reuse"

Product/Component Name	%age of Use Cases	Function Point
<Product/Component 1>	100	143
<Product/Component 2>	90	129
<Product/Component 3>	95	136
<To insert row, click on this cell and press CTRL and letter k>		
<To delete a row, click on row to be deleted and press CTRL and letter d >		
Actual Use Function Point of the Component -------		409

Fig. 7. Calculate the "actual reuse"

Around 40 and 70 reusable software assets were tracked in the years 2009 and 2010, respectively. All the certified assets were termed as "Potential Reuse". Potential Reuse measures the certified assets that are available for reuse. These assets can be used as many number of times. It can be seen that cumulative actual reuse is more than cumulative potential reuse since the actual reuse of the asset can take place more than once. Figures 8 and 9 show the reuse metrics for years 2009 and 2010 of the software development organization.

The graphs in Figures 8 and 9 show the trend of reuse in the organization. The potential reuse and actual reuse were tracked on a monthly basis to derive the graphs. It helped the organization to evaluate the software asset reuse. The first year was used to study the trend and set the target. The potential reuse increased from 553 to 1917 FP (Function Points) in the year 2009. In the next year, the potential reuse increased from 1917 to 2911 FP.

The organization set a cumulative yearly reuse target of 6000 FP. The graph depicts the actual reuse met by the development team to be 7801 FP in the year 2009 and 7185 FP in the year 2010.

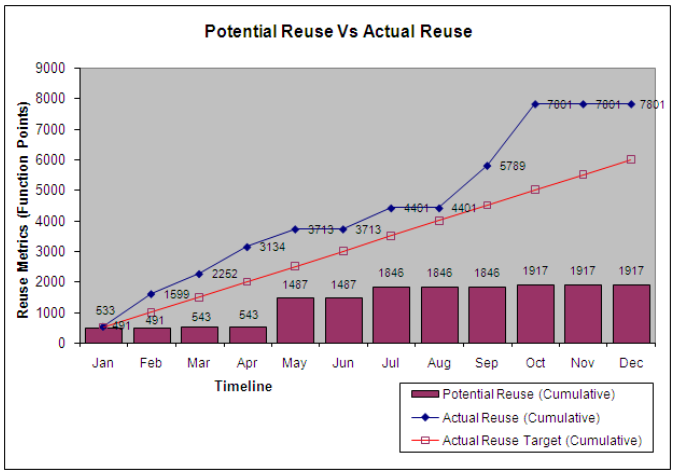


Fig. 8. Cumulative "Potential Reuse" versus "Actual Reuse" of all types of reusable assets for the year 2009

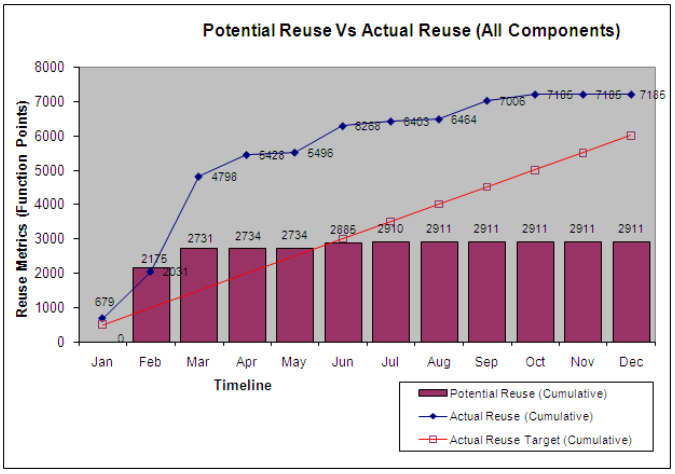


Fig. 9. Cumulative "Potential Reuse" versus "Actual Reuse" of all types of reusable assets for the year 2010

The graph in Figure 10 is a non cumulative graph of Potential Reuse versus Actual Reuse for the year 2009. Similar information is in Figure 11 for the year 2010. These graphs helped in identifying the reuse in each month clearly and helped the managers/directors to track and communicate the need of reuse to the organization. The cost saving was derived in a crude manner. First the Effort in Man Hours spent per FP was calculated by adding total number of potential reuse FP generated for six months period and dividing it by total effort spent in developing the assets. The author refers it as "crude" because the assets considered during the six months were of different technologies, some using 3 GL and others using 4 GL languages. The assets should have been categorized before calculating the cost saving. Without the asset categorization, the cost avoidance provides a rough number that can be used for management purposes. Once the effort per FP was calculated, the total actual reuse FP count was used to generate total reuse cost saving of the organization.

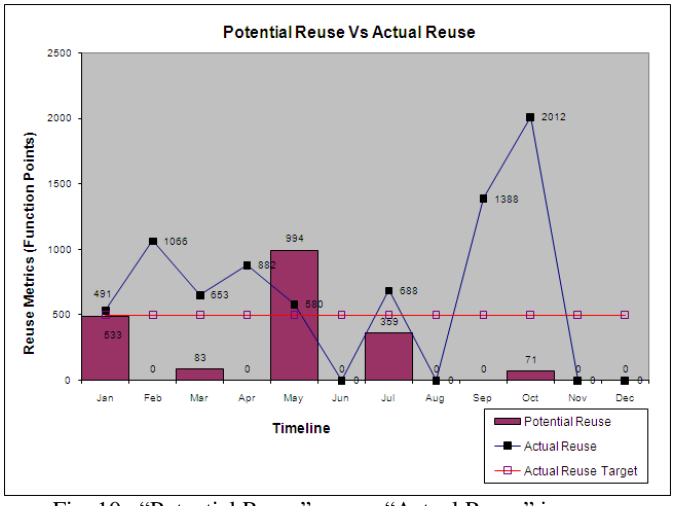


Fig. 10. "Potential Reuse" versus "Actual Reuse" in a non cumulative fashion for the year 2009

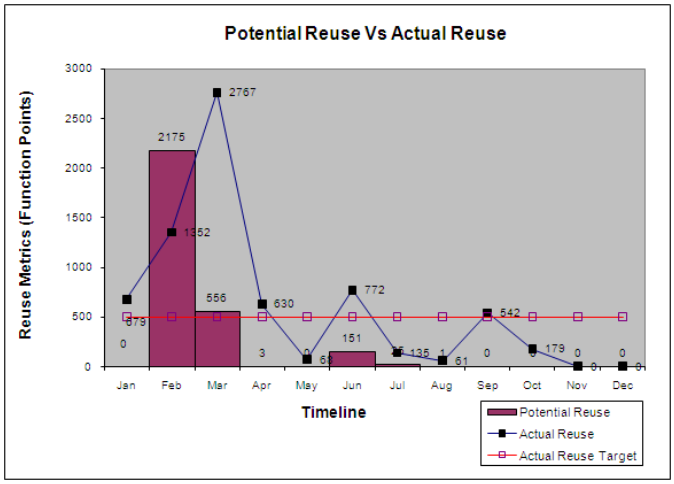


Fig. 11. "Potential Reuse" versus "Actual Reuse" in a non cumulative fashion for the year 2010

The following formula shows how the effort per function point in man hours was calculated.

$$\text{Effort per FP } (E_{FP}) = T_{Eff} / T_{PFP} \quad (3)$$

where,

T_{PFP}, Total FP of potential reuse asset created in 6 months

T_{Eff}, Total effort spent to develop the potential reuse asset

The Effort per Function Point was used to generate the total cost saving or cost avoidance. The following formula shows how the cost avoidance was calculated.

$$\text{Cost Saving for the year} = T_{AFP} * E_{FP} * C_{HR} \quad (4)$$

where,

T_{AFP}, Total FP reused in products released in a year

E_{FP}, Effort per Function Point

C_{HR}, Average cost of a developer to the organization

3 Issues in using Function Point as as Reuse Metric

This section highlights some of the issues in the implementation of the software reuse program. They are described in following subsections.

3.1 Complexity depiction

Functional Point depicts the Data flow complexity of various types of use cases. However, it does not depict the implementation algorithm complexity. So a software asset that requires a highly complex algorithm and another software asset that requires less complex processing may have similar FP count. As part of the implementation of the reuse program, the author saw many such instances.

3.1.1 Introduce McCabe & Halstead Complexity measures in Function Point calculation

The author is proposing to use McCabe and Halstead complexity measures in the calculation of Function Point to overcome this issue. It was applied on some sample software assets indentified in section II of this paper. The result showed that the new metric gave higher modified FP value for complex software assets than others. There are tools available that can auto generate McCabe and Halstead complexity measurements. A challenge is that these tools are not available for all the technology and not all solutions are affordable. This metric did produce improved results, but the company is not yet ready to make the details of this metric public.

3.2 ETL based components showed high cost savings

ETL (Extract, Transform & Load) based software asset has lots of data elements that flow across its boundaries. Based on the FP count, such assets generated high FP counts. However, the cost saving was based on a general category. Applying the cost saving formula on these assets resulted in very high cost saving. Cost savings on such assets did not reflect the reality.

3.2.1 Introduce asset categorization and refine the Effort per Function Point metrics per category

The author proposes to categorize the assets based on their functionality and technology, and to generate the Effort per Function Point for each category. The cost saving should be calculated based on the reuse category.

3.3 Subjectivity in certain areas of FP calculation

The developers/designers complained to the author about the subjectivity in GSC and the degree of influence calculation while implementing the software reuse program. This topic is also highlighted by Symons in [18] & by other authors in [1].

3.3.1 Introduce concepts of Mark II proposed by Symons in [18]

The author attempted to implement some of the Mark II enhancements on sample assets to study the impact of the calculation and the ease of use. The author filtered out some of the complexity involved in Mark II based calculations. The use of additional five GSC such as interfaces, security and privacy, user training, third party use and documentation provides more clarity in the size of the software asset. The concept introduced by authors in [1] is interesting. However, there are no standards and tools available to support this concept. It will take time and effort to implement the concept of the training database and refinement of ordinal scale conversion to absolute scale.

3.4 FP metric is not distributed at a use case level

The FP count of a software asset is not distributed at a use case level. It is a final count assigned to the whole software asset. When the actual reuse takes place, some of the use cases are reused in a product. There is not an easy way to assign the FP count for a use case and count it when it is used. In the practical approach depicted in Section II, the total FP is equally distributed to all the use cases of the software asset. This is not a precise manner to calculate the actual reuse.

3.4.1 Perform additional step to generate FP count for each use case

The author proposes to introduce additional step after step 5 to generate FP count for each use case. The derived degree of influence is applied to each use case instead of the standard way of applying the degree of influence on the total Unadjusted Function Point. The author implemented this concept in some sample assets and found it is quite reliable. There is a need to apply it for the organization and study the reliability of this step.

3.5 Developers complain about the effort spent to calculate

The developers and designers complained the author about the effort spent to calculate the metric.

3.5.1 Automate collection of metrics

The author proposes to automate the collection of metrics. However this will involve in-house time and effort to automate FP generation. The practical implementation illustrated in section II of this paper identifies that it is not too time consuming to calculate the FP. It could be the psychological aspect of developers who have to pause their daily development activities and engage in metric collection.

The effort spent approximately matches with 1 work hour per 100 FP's identified in [13]

3.6 Refine FP complexity calculation step

The complexity calculation recommended by International Function Point Users Group (IFPUG) is based on values proposed by Albrecht after the concept was used in IBM. This is referred as "step 3" and "step 4" in section II of this paper. These values have not been revised for many years. Also the ordinal ratings are applied to derive the complexity. There is a need to identify absolute ratings. This limitation is highlighted in [1], [14], and by Symons in [18]

4 Function Point & Weyuker's complexity measure analysis

If we carefully analyze the Function Point Analysis (FPA) methodology, we can see that it generates the data flow complexity of the system by assigning complexity of the use cases under various categories of EI, EQ, EO, ILF and ELF. The data flow complexity is calculated by identifying the FTR's (File Types Records) and DET's (Data Element Types) that move across the system boundaries in transactional functionality of the system, as well as RET's (Record Element Types) and DET's (Data Element Types) in Data Storage functionality of the system. The assigned complexity is further used in calculating the UFP (Unadjusted Function Point) and finally the FP (Function Point). We can safely state that Function Point reflects the complexity of the system. The classification of Function Point as a complexity measure is also mentioned in [9]. However the methodology does not analyze the full complexity of the system such as algorithm and implementation complexity.

In this section the author argues that FP is also a measure of complexity apart from a measure of size. He attempts to analyze Function Point metrics in a similar fashion that Weyuker analyzed other complexity measures such as McCabe's Cyclomatic number, Halstead's programming effort, statement count (LOC – Line of Count) and Oviedo's data flow complexity in [19]. Following are subsections elaborate the findings.

4.1 Property 1 – $(\exists P)\ (\exists Q)\ (|P| \neq |Q|)$

There exist two software assets whose Function Points are different. This has been observed in the 70 components mentioned in the previous section. Almost all the software assets had different FPs.

4.2 Property 2 – Let c be a non negative number. Then there are only finitely many programs of c complexity

If we extend the argument of Cyclomatic complexity or data flow complexity not following this property, it is equally arguable that FP does not follow this property. Since FP does not depict the implementation and algorithm complexity, there could be infinite software assets that may have complexity c defined above.

4.3 Property 3 – There are distinct programs P and Q such that |P|=|Q|

Even though the author did not find any of the 70 components that were studied having same FP, it is possible that two assets may have same FP. The functionalities provided by these two assets might be different however the formulae used to generate the FP may have same input values.

4.4 Property 4 – $(\exists P)\ (\exists Q)\ (P \equiv Q\ \&\ |P| \neq |Q|)$

Since program equivalence is an undecidable question [19], it is quite possible that that two equivalent assets providing similar functionality have different FP values.

4.5 Property 5 – $(\forall P)\ (\forall Q)\ (P \leq |P;Q|\ \&\ |Q| \leq |P;Q|)$

This property is not valid for FP metrics in all cases. When two assets are combined to give a single asset, it is not always necessary that the FP of the combined asset will be greater than or equal to the original assets. This is due to the fact that the use cases will change, either increase or decrease, after combining two assets. However, since the FP is directly dependent on the use case and its data flow, the inequalities in this property are true.

4.6 Property 6a – $(\exists P)(\exists Q)(\ \exists R)(|P| = |Q|\ \&\ |P;R| \neq |Q;R|)$; Property 6b – $(\exists P)\ (\exists Q)(\ \exists R)(|P| = |Q|\ \&\ |R;P| \neq |R;Q|)$

If we consider three software assets, two assets have the same FP and the third asset is combined to the two assets, it is quite likely that the FP will be different in the combined assets. This is because once the asset is combined the use cases will change, either increase or decrease, hence the combined FP can be lesser or greater. So this property follows for Function Point complexity measure.

4.7 Property 7 – There are program bodies P and Q such that Q is formed by permuting the order of the statements of P, and $|P| \neq |Q|$

Function Point does not depend on the order of the statements of a reusable asset. It is dependent on the supported use case and the data flow for these cases between the system and the interacting world. The reordering of assets will not change the use case and the functionality. Hence FP does not follow this property.

4.8 Property 8 – If P is a renaming of Q, then |P| = |Q|

Function Point does not change if an asset is renamed. This property holds true for FP.

4.9 Property 9 – (∃P)(∃Q)(|P|+|Q| < |P;Q|)

This property holds true for Function Point. For some of the assets the sum of the FP of two assets will be lesser than the FP of the combined asset. The property says that this is applicable for some assets. For all assets this might not hold true. It has been observed that for some asses after adding additional functionality and refactoring the code, the total Function Point value decreased.

Property Number	Statement Count	Cyclomatic Number	Halstead Effort	Data Flow Complexity	Function Points
1	YES	YES	YES	YES	YES
2	YES	NO	YES	NO	NO
3	YES	YES	YES	YES	YES
4	YES	YES	YES	YES	YES
5	YES	YES	NO	NO	NO
6	NO	NO	YES	YES	YES
7	NO	NO	NO	YES	NO
8	NO	YES	YES	YES	YES
9	NO	NO	YES	YES	YES

Fig. 11. Summary of the Weyuker Property analysis with the Function Point metric included

A complexity measure for which all the properties of Weyuker hold does not indicate that it is the best complexity measure. One should understand the intent of the measure and apply adequate weights to these properties before comparing the measures. For example the comparison shown in Figure 11 does not mean that Data Flow complexity measure is better than Function Point simply because the former measure satisfies more properties.

5 Conclusions

Implementation of an efficient software asset reuse program is very important to a software development organization to be competitive. The program allows the organization to effectively manage their resources and increase the productivity of the associates. There is a need for the development society to standardize the method and metrics for software reuse. It will be another milestone for the software development methodology to be qualified as an engineering discipline.

6 Acknowledgement

Johns T. Joseph is grateful to Professor Dr. Austin C. Melton for the guidance, feedback and critiques provided while working on this paper. He also thanks the classmates for attending his presentation, and providing valuable and critical feedback. He acknowledges his family in providing indispensable support that enabled him to spend time and effort to research on the topic.

7 References

[1] M. A. Al-Hajri, A. A. A. Ghani, M. N. Sulaiman, M. H. Selamat, "Modification of standard Function Point complexity weights system", The Journal of Systems and Software 74, 2005

[2] A. J. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, Vol. SE-9, No. 6, November 1983

[3] J. S. Collofello, S. N. Woodfield, and N. E. Gibbs, "Software Productivity Measurement", National Computer Conference, 1983

[4] C. J. Dale, "Software Productivity Metrics who needs them?",EC2 Publications, 1992

[5] J. J. Dolado, "A study of relationships among Albrecht and Mark II Function Points, Line of Codes 4GL and Effort", J. Systems Software, 1997

[6] N. Fenton and Austin Melton, "Measurement Theory and Software Measurement", in A. Melton, editor, Software Measurement: Understanding Software Engineering, International Thomson Computer Press, 1995

[7] N. Fenton, "Software Measurement", IEEE Transactions on Software Engineering", Vol. 20, No. 3, March 1994

[8] W. Frakes and C. Terry, "Software Reuse: Metrics and Models", ACM Computing Surveys, Vol. 28, No. 2, June 1996

[9] S. Furey, "Why we should use Function Points", CounterPoint, IEEE March/April 1997

[10] T. Hall and N. Fenton, "Implementing effective software metrics program", IEEE Software 1997

[11] D. R. Jeffrey, G. C. Low, and M. Barnes, "A Comparison of Function Point Counting Techniques", IEEE Transactions on Software Engineering, Vol. 19, No. 5, May 1993

[12] C.F. Kemerer and B.S. Porter, "Improving the reliability of Function Point Measurement: An empirical study", IEE Transactions on Software Engineering, Vol. 18, No. 11, November 1992

[13] C. F. Keremer, "Reliability of Function Points Measurement – A Field Experiment", Communication of the ACM, Vol 36, No. 2, February 1993

[14] B. Kitchenham, "The Problem with Function Points", CounterPoint, IEEE Software 1997

[15] G. C. Low and D. R. Jeffery, "Function Points in the Estimation and Evaluation of the Software Process", IEEE Transactions on Software Engineering, Vol. 16, No. 1, January 1990

[16] T. J. McCabe, "A Complexity Measure" Transactions on Software Engineering, Vol. SE-2, No. 4, December 1976

[17] L. M. Ott. The Early Days of Software Metrics: Looking Back After 20 Years, in A. Melton, editor, Software Measurement: Understanding Software Engineering, International Thomson Computer Press, 1995

[18] C. R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Vol 14, No. 1, January 1988

[19] E. J. Weyuker, "Evaluating Software Complexity Measures", IEEE Transactions on Software Engineering, Vol. 14, No. 9, September 1988

[20] A. C. Melton, D. A. Gustafson, J. M. Bieman and A. L. Baker, "A Mathematical perspective for software measures research", IEE/BCS Software Engineering Journal, 1990

[21] S. AMBLER, "The Strategic Reuse Discipline: Scaling Agile Software Development", http://www.enterpriseunifiedprocess.com/essays/strategicReuse.html

What Software Measurement Can Learn from Classical Measurement and Measurement Theory

Amruta Sakhrani[1] **and Austin Melton**[2]
[1]Department of Computer Science, Kent State University, Kent, Ohio 44240, USA
[2]Departments of Computer Science and Mathematical Sciences, Kent State University, Kent, Ohio 44242, USA

Abstract— *Though software measurement is relatively new - seriously beginning in the 1970s, measurement is well developed in many disciplines, and measurement theory is itself a well developed discipline. Probably because software engineering has developed so very quickly and because managers understand the value of measurement, the field of software measurement has exploded in the last three decades. However, as can be expected in an explosion, the development of software measurement has not always progressed in the most productive manner. In this paper, we look at classical measurement and measurement theory to see some ways for possibly improving software measurement. We believe there are, at least, four things that we can borrow from measurement and measurement theory to improve software measurement. These are the transition from quality to quantity, the importance and need for theories for software metrics, a system of units for software metrics, and the standardization of software metrics and software measurement practices.*

Keywords: Measurement, measurement theory, models, software engineering, software metrics, theories, units

1. Introduction

Software measurement began at least by the time people started counting lines of code. As an organized field of study and research, software measurement, which was first called software science, began in the early 1970s with Maurice Halstead and his students and co-workers at Purdue University. Though software measurement involves measurement as the term is commonly used, it is not measurement in the classical sense; please see [1]. Further, although classical measurement theory ideas have been used in software measurement, classical measurement theory is not an appropriate theory for software measurement. However, as classical measurement and software measurement do have elements of commonality, e.g., the goal of quantification, it is worthwhile to see what software measurement can learn from classical measurement.

In this paper, we are not trying to develop a software measurement theory, but we do look at four ideas/practices which are part of classical measurement theory and which we believe would improve software measurement if these ideas/practices were implemented in software measurement. Thus, when a satisfactory theory for software measurement is developed, we would hope that the theory supports these ideas and practices. These ideas and practices are

- understanding the differences between quality and quantity when discussing an attribute of a software artifact or process and understanding the transition in moving from quality to quantity,
- understanding the value of a theory and a model for each software metric,
- developing a system of units for software metrics, and
- standardizing software metrics and software measurement activities.

2. Quality and Quantity

In discussions about measurement, one should consider Lord Kelvin's well known statement: [2]

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind: it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of *science*.

This statement clearly supports measurement efforts, but the quote is often used improperly.It does not say that whenever something is expressed in numbers, it is understood. A characteristic orattribute of a set of entities isunderstood if the numbers reflect true measurement, i.e.,if the numbers truly reflect the quality of the characteristic in the entities.

Another interesting quote comes from sociologist William Bruce Cameron. [3][1]

> It would be nice if all of the data which sociologists require could be enumerated because then we could run them through IBM machines and draw charts as the economists do. However, not

[1]A variation of the last sentence in this quote is often attributed to AlbertEinstein or is reported to have been on a sign hanging in his Princeton office. A Googlesearch gives several websites supporting these two attributions.

everything that can be counted counts, and not everything that counts can be counted.

Both in the historical development and logical structure of scientific knowledge, the formulation of a theoretical concept or construct, that defines a quality, precedes the development of measurement procedures and scales. Said differently, before a quality can be quantified, it must be understood in ways that allow for quantification. For example, the concept of heat as a theoretical construct interpreting the multitude of phenomena involving warmth was necessary before one could conceive of and construct a thermometer.

In the development and maintenance of software there is so much at stake financially and otherwise, that it is easy to understand why managers want to quantify everything about the resources, products, and processes. However, these needs for quantification must to be tempered with an understanding of the difficulties in quantifying attributes and in an acknowledgement of the potential serious problems which can follow from inaccurate quantification. Also, there is the possibility that some things associated with software may not be quantifiable or quantifiable in only coarse ways.

For example, there is difficulty in the measurement of such qualities as beauty. The existence and meaningful use of the word beauty indicates the usefulness of the concept. However, there is no objective rule (please see the next paragraph) for classifying some observable aspect of beauty. Similarly, there are no objective empirical relations such as indistinguishability or precedence in respect of beauty. The basis for objective measurement of beauty is thus absent from the outset.

The concept of a quality is formed as an objective rule for the classification of an empirically observable aspect of objects or entities of a single set. If one works under the representation theory of measurement, the distinction between quality and quantity and the transition from quality to quantity are obvious. [4], [5] In the representation theory of measurement, one begins with an entity set and an aspect or characteristic of the entities in the set. If the "objective rule for classification" mentioned above allows an observer to form empirical relations on the set of entities and if there exists a function from the set of entities and the collection of empirical relations to a set of mathematical objects (e.g., the set of non-negative real numbers) and a collection of mathematical relations on the set of mathematical objects such that the function fully and faithfully depicts the empirical relations by the mathematical relations, then one has moved from quality to quantity with the quantity being represented by the mathematical set of objects and the mathematical relations. An important point of this discussion is quantifying an aspect or characteristic of the elements of an entity set is a truly significant activity. In fact, this is the intellectuallychallenging aspect of measurement; this is when the understanding which precedes taking measurements occurs. Determining the measurement value for a given entity is simply a matter of applying the tool or tools which are developed in this quantification step. Of course, this "step" may be a lengthly process.

Of course, it is not the case that methods from the representation theory of measurement need to be used to make the transition from quality to quantity, but the representation theory clearly shows the transition. Implicit in representation theory is the idea that the qualitative appreciation or understanding of the characteristic in question must produce "objective" empirical relations, i.e., the classification of the entities with respect to the characteristic being observed needs to be independent of the observer. This itself implies a willingness of observers to follow guidelines and standards established for the classification. Interestingly, though this paper is divided into four ideas and practices, these ideas and practices are intricately interwoven.

3. Theories and Models for Metrics

In this section, we are not using "theory" in the sense of a theory for a field or discipline like measurement theory. We are using the term with metrics in the sense of developing an understanding of what a proposed metric is supposed to do and providing a justification for how the metric is defined. Additionally, by using the theory for a metric, one should be able to explain and justify a model for the metric.

For example, if McCabe's cyclomatic metric were being defined using these theory and model ideas, one would explain why control flow is an important aspect of the complexity of a program and why/how the control-flow complexity is related to the number of linearly independent execution paths through a program or the number of decisions in the program. Control-flow graphs would constitute a model for the McCabe cyclomatic metric. [6]

The development of a theory and a model for each metric has a number of benefits. By explaining what a metric is trying to do and why what it measures is meaningful, other researchers and users are able to meaningfully critic the metric for possible improvements and to propose potential uses. Having these theories and models for each metric would improve the use of metrics and improve the field of software measurement. For example, in Weyuker's well known paper on properties of metrics, she unfairly criticizes McCabe's metric because it does not measure the complexity due to the amount of computation performed even if the computation does not increase control-flow complexity. [7] Theories and models for metrics would help prevent unjustified criticisms and would help researchers and users better understand software metrics in particular and software measurement in general.

1) Models provide representation of some aspect of the real-world system of interest.
2) They enable us to investigate the real situation without needing to actually produce it or modify an existing situation.

3) They provide means to motor ingenuity, that is, they give us a chance to imagine the working of the real system.

The importance of theories and models is even more significant when a metric uses internal measurement values such as control-flow complexity or lines of code to measure an external characteristic such as cost or person-hours to complete a project. In cases such as these, the theory for the metric needs to establish the connections between the internal characteristics and the external one. It is not sufficient to simply state, for example, that as the control-flow complexity increases it naturally follows that the maintenance costs will increase. This understanding may, of course, be a starting point, but it is not sufficient by itself to justify determining or estimating maintenance costs using control-flow complexity. The needed theory is the justification for how and how much the internal characteristics influence the external characteristic which is the object or goal of the measurement and estimation, and the theory is thus the basis and the justification of the model which gives the details of the measurement or prediction formula. If another researcher or a user has questions about the formula, s/he should be able read the corresponding theory to understand the formula details. This theory-model connection allows other researchers to build upon and improve the research already done. It may be the case that there is so little follow-on or extended research on individual software metrics because metric models (which may be the metric definitions) seem to rarely have adequate supporting theories. The theories would be more easily developable if the transition from quality to quantity were more thoroughly done because then we would more completely understand what is being measured and the metric itself.

4. Measurement Units

One of the most striking differences between measurement in most disciplines and in software engineering is the lack of units in software measurement. Probably for most people, when we think of measurements, we think of numbers with units, e.g., 68 kilograms or 163 centimeters. If someone told us that a measurement value is 27, we would probably immediately ask "27 what?"

There are, of course, reasons for these missing units. From a technical point of view and thinking in terms of the representation theory of measurement which has five levels or scales of measurement (classification, ordinal, interval, ratio, and absolute) [4], [5], a metric must be of the interval, ratio, or absolute type before units can be meaningfully defined. Unfortunately, many software metrics are of the ordinal type. The measurement values from ordinal metrics are only comparatively meaningful. Thus, if ordinal metric M is defined on an entity set and if s and t are entities, then $M(s) < M(t)$ only means that entity s has less of the characteristic being measured than does entity t. The difference between $M(s)$ and $M(t)$, i.e., $M(t) - M(s)$ has no meaning with respect to how much more of the characteristic is in t than is in s.

A second reason for the lack of units in software measurement is that, unfortunately, most of our measurement situations are not well enough understood to know what the units should be. This may be due, in part, to software engineering being a relatively new discipline or because software engineering is developing and changing so quickly. Whatever the reasons, the lack of units hinders the development of software measurement.

As many of us learned in physics classes, one can perform type checks on calculations using units. For example, if one is working with speed, time, and distance and one is trying to calculate how far a car travels given its speed and the time of travel. One knows the answer should be a distance. Thus, when one multiples a speed in terms of kilometers per hour times time in terms of hours, one gets a distance, namely, kilometers. Unfortunately, this kind of type checking of calculations is usually not possible in software measurement because we have not developed a system of units, i.e., we have not developed a type system for our measurements.

The problem with this lack of units is not simply one of type checking calculations. There are many things in software engineering which are important and which we want and need to measure but which must be measured indirectly. Thus, researchers, as they try to define these important indirect measurements,know that they will need to measure other characteristics and then combine the other measurementsfor the indirect measurement they are seeking. If we knew the units or types of our measurements,this could be of significant help to researchers as they try to develop these indirect metrics.The benefits of knowing how to combine known measurements to obtain needed indirect measurements would bemore easily achieved if we had good theories and models for our metrics.

Although having units for our metrics would be a wonderful addition to software measurement, itis probably too much to ask or expect in the near future for researchers to determine precise units for most software metrics. However,maybe there is good compromise or middle ground

1) which is between where we are now without unitsand where we could be with units,
2) which we can achieve in the near future, and
3) which will have many of the benefits of having units.

The SI (The International System of Units) system of units has seven base quantities: length, mass, time, electric current, thermodynamic temperature, amount of substance and luminous intensity. Other quantities, which are called derived quantities, are defined in terms of the base quantities. [8]

We could decide on base quantities for software measurement and then begin discussions regardingderived quantities.

Further, if we wouldd determine the base and derived quantities for existing metrics, then, for example, instead of saying ametric's value for a given entity is 27, we could say 27 length, where "length" represents the type of themeasurement value. Another benefit of giving measurement types is we could also do this for ordinal and even classification metrics.

As a starting point in the discussion to determine the base quantities, we could begin with the measurement categories given by Briand, Morasca, andBasili in [9]. Their categories are size, length, complexity, cohesion, and coupling.

These activities in determining base and derived quantities would help considerably in thestandardization of software measurement.

5. Standardization in Software Measurement

Measurement in the physical sciences has a long history. It has been developed, evaluated, documented, and used extensively. Similar usage is followed in almost all countries. What can be said about software measurement? Why do we not have standardization of software terms and practices? It may be because software activities started quite recently as compared to physical science measurement activities, and in that short time frame, many terms and methods for performing various software tasks have been discovered. Hence, less time relative to developments has been devoted to the evaluation and analysis of these discoveries and activities. Also, we have essentially no physical laws and very few agreed upon practices for guiding the design and definition of software metrics. Today, when a company that develops software decides to establish a metrics program, it is not uncommon for the company to establish its own measuring techniques and terms. In such cases, the customer (end user) and the software developer may not even use the same practices and terms in establishing and describing the requirements and quality of the software. Standardization in software measurement and software engineering could, therefore, be very beneficial for the end users as well as for researchers.

No doubt, standardization in software measurement will eventually occur. For the field of software engineering to become recognized as an established discipline, it seems that standardization must occur. Thus, on the one hand, it does seem that standardization in software measurement would be a natural outgrowth of the maturing of software engineering.[2] On the other hand, if we promote standardization in software measurement, we will hasten the maturing of software engineering.

As stated earlier, the four ideas and practices which comprise this paper are themselves interwoven. As we establish good methods for transitioning from quality to quantity, as we develop good practices for developing appropriate theories and models for our software metrics, and as we move towards establishing units for our metrics, we will simultaneously be furthering the standardization of software measurement.

[2]It may seem that the authors sometimes interchange the expressions "software engineering" and "software measurement." We hope that is not the case. We do, however, believe that the two concepts are inseparably interwoven in that true and accurate software measurement will only occur when the practices and activities of software engineering are accurately understood. Also, when these practices and activities are accurately understood, then we will be able to have true and accurate software measurement.

6. Conclusions

In this paper, we have looked at four classical measurement and measurement theory ideas and practices which we believe would be beneficial if applied to software measurement. Interestingly, though these fours ideas/practices seem different and distinct, they are, in fact, closely interrelated. As one understands a quality aspect or characteristic of the entities in an entity set, so that that aspect may be quantified, one is simultaneously laying the groundwork developing the theory for the metric which measures that aspect or characteristic. Further, as one develops a model or models supporting the theory, then one is laying the groundwork for determining the unit or unit type for the metric. Standardization is, of course, inherently interwoven with the other three because in the process of standardizing software measurement practices we will, in time, require theories, models, and units for our software metrics and we will develop procedures and general guidelines for theory and model development and for defining units. Thus, the more standardization we have the easier it will be to develop theories and models for metrics and to define units, and underlying the theory development is an understanding of the transition from quality to quantity.

In this paper, we have not tried to develop a foundation for software measurement, i.e., we have not tried to develop a software measurement theory. However, we have discussed ideas which we believe a software measurement theory should promote. Thus, we are promoting guidelines or properties which a software measurement theory should have and/or should promote. Thus, when an adequate software measurement theory is developed it should

1) describe how to transition from qualitative evaluations to quantitative evaluations,
2) promote and maybe even describe how theories and models for software metrics may be developed and validated,
3) promote or even mandate the use of units and show how units may be combined for derived quantities, and
4) promote the establishment of software measurement standards.

References

[1] T. W. Reese, "The application of the theory of physical measurement to the measurement of psychological magnitudes, with three experimental examples," *Psychological Monographs*, vol. 55, no. 3, 1943.

[2] L. Kelvin, *Popular Lectures and Addresses*, vol. 1, 1889.

[3] W. B. Cameron, *Informal Sociology: A Casual Introduction to Sociological Thinking*. New York: Random House, 1963.

[4] N. E. Fenton and S. L. Pfleeger, *Software Metrics - A Rigorous Approach*. Thomson Publisher, 1998.

[5] H. Zuse, *A Framework of Software Measurement*. de Gruyter, 1998.

[6] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Dec. 1976.

[7] E. J. Weyuker, "Evaluating software complexity measures," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1357–1365, Sept. 1988.

[8] P. H. Sydenham, Ed., *Handbook of Measurement Science*. J. Wiley, 1982, vol. 1.

[9] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based software engineering measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, pp. 68–86, January 1996.

SESSION

NOVEL APPLICATIONS + REQUIREMENTS ENGINEERING + PRODUCTIVITY + SECURITY + WORKFLOWS + VALIDATION + CERTIFICATION + TESTING + OO + CLOUD COMPUTING + RISK MANAGEMENT

Chair(s)

Prof. Hamid R. Arabnia

Local Independence Transformation and Its Application to Removing Nonduplicate Statements in Code Clones

Chung Yung, Yen-Chang Lai, and Qin-Xin Tu
Department of Computer Science and Information Engineering
National Dong Hwa University, Hualien, Taiwan, R.O.C.

Abstract— *This paper proposes a new program transformation technique and describes its application to removing nonduplicate statements in code clones while preserving the semantics of the program. Allowing nonduplicate statements in code clones helps in finding larger code clones. However, the nonduplicate statements in code clones may induce unexpected difficulty and complexity when we analyze the software systems. In this paper, we propose a new technique of program transformation, called local independence transformation. The transformation analyzes the definitions of variables, and replaces the uses of the variables with their definitions in terms of constants and nonlocal variables, when possible. With the proposed transformation, some of the nonduplicate statements in code clones may be transformed into locally independent statements and hence may be moved out of the code clones while preserving the semantics of the program. Based on the new technique, we develop a three-step framework for moving the nonduplicate statements out of code clones. In the experiments on the wget source code, our approach successfully moves more than 77.9% of the nonduplicate statements out of the code clones.*

1. Introduction

This paper proposes a new technique of program transformation and describes its application to removing the nonduplicate statements out of the clones while preserving the semantics of the program. The industry reports that large-scale computer programs typically contain 10 to 25% duplicate code, called as *clones* [1]. The code clones can be a problem when the software engineers maintain the software systems [2], [3]. The clone detection techniques are useful to make sure that bugs fixed in one copy get fixed in the others and to refactor the code and eliminate redundancy [4], [5], [6].

There are many techniques proposed for detecting consecutive clones [3], [5], [7], [8], [9], [10], [11], [12], [13]. According to Bellon et al. [14], the clones detected are classified into three types:

- *Type 1* is an exact copy without modifications,
- *Type 2* is a syntactically identical copy; only variable, type, or function identifiers were changed, and
- *Type 3* is a copy with further modifications; statements wer changed, added, or removed.

On the other hand, according to Rysselberghe and Demeyer [15], the clone detection techniques can be divided into three categories: the *string based* techniques [7], [9], the *token based* techniques [3], [10], and the *abstract-syntax-tree(AST) based* techniques [5], [16], [17]. The string based clone detection divides the program into a number of strings, such as lines, and all the strings are compared with each other textually [7]. The token based clone detection transforms the program into a token stream and then search the similar token sequence using suffix tree [3]. The AST based clone detection finds similar subtrees after building a parse tree [5].

When the programmers develop code by copying and modifying sections of code, the modification usually splits a large consecutive clone into two or more small consecutive clones. This leads to the detection of similar code clones; in other words, the code clones with nonduplicate statements allowed [5], [18]. However, the nonduplicate statements in code clones may induce unexpected difficulty and complexity when we maintain the software systems. The semantics-preserving procedure extraction technique proposed by Komondoor and Horwitz moves a set of selected statements together so that they become "extractable" while preserving the semantics [19], [20]. They analyze the data dependence among statements and try to move the non-duplicate statements out of the clones when possible. If there is any non-duplicate statements that cannot be moved out, a considerable overhead is induced when the clone is extracted into a procedure. The overhead includes introducing additional procedure arguments and transforming the non-duplicate statements into conditional statements in the extracted procedure.

This motivates our investigation on the program transformations needed for moving more non-duplicate statements out of the clones when possible. We analyze the data dependence of the variables in the program, and observe that program transformations help in exploring more non-duplicate statements that can be moved out of the code clones. The basic idea can be demonstrated by the example shown in Figure 1.

In Figure 1, code blocks (a) and (b) have two identical statements, marked with "++", that may be considered as clones. However, there is an additional statement (2) in (b). We note that statement (2) cannot be moved out of the clone since the variable a is defined in statement (1) and the variable b is used in statement(3). We observe that the code block (b) can be transformed into (c) while preserving the semantics. Extracting the procedure out of the marked statements from code blocks (a) and (c) reduces the overhead

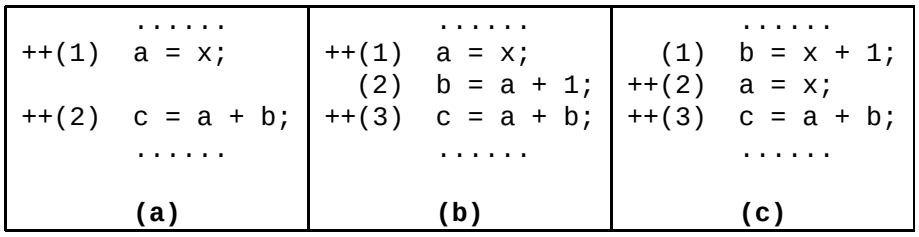


Fig. 1
A MOTIVATING EXAMPLE

of refactoring the code.

As such, we define a new program transformation called *local independence transformation*. Based on the transformation and code motion, we propose a new framework for removing the nonduplicate statements in the code clones. We implement the proposed framework and integrate into the gcc compiler and experiment on the wget source code. As a result, 77.9% of the nonduplicate statements may be moved out by our approach.

This paper is organized as follows. The following section introduces the preliminary material of this research. Section 3 defines the data dependence properties that we used in the program transformation. The proposed framework is described in Section 4. Section 5 analyzes our framework and shows the correctness. In section 6, we describe our implementation and present the experiment results. At last is a brief conclusion.

2. Preliminaries

In this section, we first give a brief review at the three categories of clone detection techniques [15]. And then, we briefly describe the semantics-preserving procedure extraction technique proposed by Komondoor and Horwitz [19].

2.1 Categories of Clone Detection Techniques

According to Rysselberghe and Demeyer, the techniques of clone detection can be classified into three categories.

- *String based clone detection*, which divides the program into a number of strings which are compared to each other,
- *Token based clone detection*, which divides the program into a stream of tokens and then searches for similar token sequences, and
- *Abstract syntax tree (AST) based clone detection*, which builds a complete AST and then performs pattern matching to find similar subtrees.

Ducasse et al. [7] propose a string based textual approach of clone detection, which is based on simple string matching. They textually report and synthesize the duplication found. Their result is visualized using scatter-plots, which are helpful means in analyzing duplication.

The token based clone detection tokenizes the program with lexical analysis while not parsing the program [4], [10]. They search the token sequence for identical patterns in the token sequence. Hence, the clone detection problem is reduced into a pattern matching problem.

Baker's token based clone detection transforms the token sequence into a parameterized string, or a *p-string*. The p-strings are compared with a parameterized matching called *p-match*, which finds the common suffix sequences of the p-strings [3]. Note that in the p-matched strings, there is a one-to-one function that maps the set of parameters in one section onto the set of parameters in the other.

The abstract-syntax-tree (AST) based clone detection compares each subtree in the same hash partition through tree matching after analyzing the syntax tree [5], [17]. The approach proposed by Baxter et al. compares each subtree of the abstract syntax tree by computing the similarity [5]. If there are two subtrees whose computed similarity exceeds a threshold, the subtrees are called clones.

They compute the similarity between two subtrees by following formula:

$$\text{Similarity} = \frac{2 * S}{2 * S + L + R}$$

where

S = number of shared nodes,
L = number of different nodes in subtree 1, and
R = number of different nodes in subtree 2.

Their basic algorithm finds single-subtree clones, but the subtree-sequence clones could not be detected. The basic algorithm is straightforward. It hashes all subtrees into buckets while similar subtrees are hashed into the same bucket. Then, it compares every pair of subtrees located in the same bucket. If the similarity is higher than the specified threshold, the pair of subtrees is added to the clone list. In fact, each clone detected by basic algorithm is a single subtree which usually represents a single statement in the source program.

For detecting the statement sequence clones, Baxter et al. build a list structure where each list is associated with a statement sequence in the program. The hash codes of each subtree element in the associated sequence are stored in the list structure. The sequence detection algorithm compares each pair of subtrees containing nodes looking for the maximum length of possible sequencing that encompasses a clone.

Note that the computation complexity of the clone detection algorithms proposed by Baxter et al. are high because of the complex tree matching algorithm used in their approach.

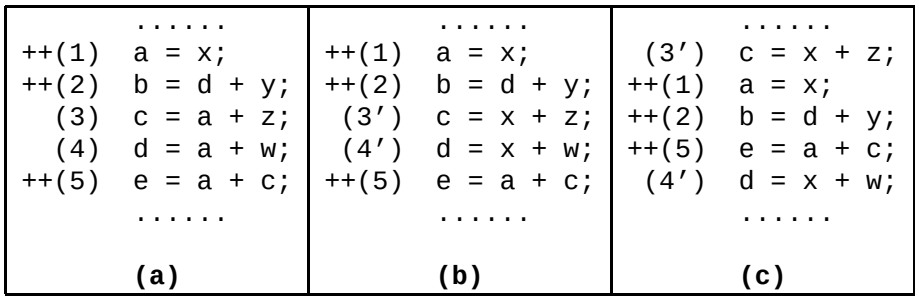


Fig. 2
AN EXAMPLE OF LOCAL INDEPENDENCE AND VIRTUALLY LOCAL INDEPENDENCE

2.2 Semantics-Preserving Procedure Extraction

Komondoor and Horwitz propose a semantics-preserving procedure extraction that moves the duplicate statement together so that they form a sequence that can be extracted into a procedure, if the duplicate statements are not contiguous [19], [18], [20]. They assume that programs are represented using a set of control-flow graphs (CFGs) [21], one for each procedure. They also assume the following inputs to their algorithm:

1) $\mathcal{P}$, the control-flow graph of a procedure, and
2) the set $\mathcal{M}$ of nodes in $\mathcal{P}$ that have been chosen for extraction.

The goal of their algorithm is to produce a CFG $\mathcal{P}_{\mathcal{M}}$ that includes exactly the same node as $\mathcal{P}$ so that:

- the nodes in $\mathcal{M}$ are extractable from $\mathcal{P}_{\mathcal{M}}$, and
- $\mathcal{P}_{\mathcal{M}}$ is semantically equivalent to $\mathcal{P}$.

The algorithm proposed by Komondoor and Horwitz consists of the following steps [20]:

- Step 1: Identify the code region to be transformed.
- Step 2: Determine a set of ordering constraints, including the following:
  - *Data-dependence constraints*: The data-dependence constraint $m \leq n$ is generated for each pair of nodes m, n such that there is a flow, def-order, anti, or output dependence from m to n.
  - *Loop-structure constraints*: The loop-structure constraint $m = n$ is generated for each pair of node n, m such that both are part of a strongly-connected component in the clone.
  - *Control-dependence constraints*: For each node n in the clone and for each control ancestor p of n in the clone, the control-dependence constraint $m \Rightarrow p$ is generated.
  - *Extended constraints*: The generated data-dependence, loop-structure, control-dependence constraints are used for generating extended constraints according to the following rules.
    1) $p \leq q$ and $q = n$ generates a new constraint $p \leq n$.
    2) $n \Rightarrow p$ and $a \leq p$, $b = p$, and $p \leq c$ generates $a \leq n$, $b = n$, and $p \leq c$.
    3) if j is an exiting jump and n is an antecedent of j, then for each constraint $j \leq m$ or $j = m$ generate a new constraint $n \leq m$.
- Step 3: Promote all unmarked node n for which one of the following conditions holds:
  1) There is a constraint $n = m$ for som marked node m, or
  2) There are constraints $m_1 \leq n$ and $n \leq m_2$ for some marked node m_1 and m_2.
- Step 4: Partition the nodes in the region into three "bucket": *before*, *marked*, and *after*. Place all marked nodes in the marked bucket, and the unmarked nodes in before and after buckets.
- Step 5: This step does some final processing of non-exiting gotos and returns, as well as exiting jumps.
- Step 6: The three buckets are converted to code blocks.

Komondoor and Horwitz show that the converted code is semantically equivalent to the original code [19], [20].

3. Local Independence Transformation

In this section, we introduce the properties of local independence and virtually local independence, and their application to local independence transformation while preserving the semantics of the program.

We first define the local independence property of a variable as follows.

Definition (local independence of variables): A variable x in a code clone $\mathcal{C}$ is *locally independent* if one of the following conditions holds.

- x is known to be a constant, or
- x's definition is not changed in $\mathcal{C}$. □

Extending from the local independence property of variables, we give the definition of local independence property of statements.

Definition (local independence of statements): A statement s in a code clone $\mathcal{C}$ is *locally independent* if both of the following conditions hold.

- All the variables used in s are locally independent, and
- All the variables defined in s are not used in $\mathcal{C}$. □

Note that the local independence property of statements is a strong property. Concerning with code motion, we are more interested in two other properties of statements; namely, the backward local independence property and the forward local independence property. Their definitions are given as follows.

Definition (backward local independence of statements): A statement s at a program point p in a code clone $\mathcal{C}$ is *backward locally independent* if both of the following conditions hold.

- For each variable v used in s, the definition of v is not changed from the beginning of $\mathcal{C}$ to p, and
- For each variable v defined in s, v is not used from the beginning of $\mathcal{C}$ to p. □

Definition (forward local independence of statements): A statement s at a program point p in a code clone $\mathcal{C}$ is *forward locally independent* if both of the following conditions hold.

- For each variable v used in s, the definition of v is not changed from p to the end of $\mathcal{C}$, and
- For each variable v defined in s, v is not used from p to the end of $\mathcal{C}$. □

It is clear that for a code clone $\mathcal{C}$, forward locally independent statements may be moved to the beginning of $\mathcal{C}$ and backward locally independent statements may be moved to the end of the $\mathcal{C}$ with the semantics of $\mathcal{C}$ preserved. If a statement s is neither backward locally independent nor forward locally independent, s is *locally dependent*.

Now, we define a new program transformation called local independence transformation as follows.

Definition (local independence transformation): A *local independence transformation* transforms a locally dependent statement s into a statement s', denoted as $s' = \mathcal{T}_v^r(s)$, by repeatedly replacing a variable v used in s with the uniquely immediate reaching definition of v at statement r, and s' is either backward locally independent or forward locally independent. □

For example, in the code clone shown in Figure 2(a), duplicate statements are marked with "++", and the unmarked statements are nonduplicate. Note that we cannot move the statements (3) and (4) out of the clone because of the data dependence. A local independence transformation on statement (3) gets statement (3'), and a local independence transformation on statement (4) gets statement (4'), as shown in Figure 2(b). It is clear that the transformations preserve the semantics of the clone. Note that statement (3') is backward locally independent, and statement (4') is forward locally independent. Hence, we get an equivalent code clone shown in Figure 2(c), which can be used for procedure extraction with the overhead reduced.

4. Removing Nonduplicate Statements

This section describes our framework for removing nonduplicate statements in code clones using the techniques of local independence transformation and code motion.

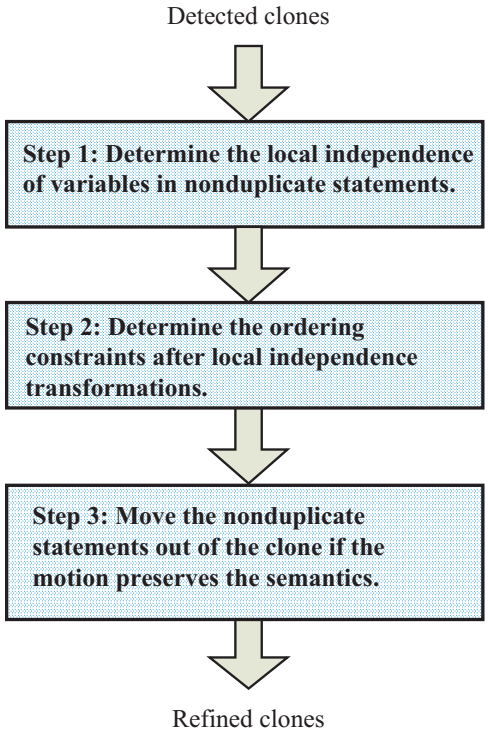


Fig. 3
THE FRAMEWORK FOR REFINING LARGE CLONES

When the programmers develop code by copying and modifying sections of code, the modification usually splits a large consecutive clone into two or more small consecutive clones. Here, we are interested in the large clones with nonduplicate statements in them. However, the nonduplicate statements in code clones may induce unexpected difficulty and complexity when we maintain the software systems. We propose a three-step framework that moves the nonduplicate statements out of the code clones using the techniques of local independence transformation and code motion.

Before applying our framework, we assume that the clones in the program are already identified with duplicate statements marked and nonduplicate statements unmarked. This assumption can easily be realized by straightforward modification on any of the consecutive clone detection techniques described in Section 2.

As shown in Figure 3, our framework consists of three steps:

- Step 1: Determine the local independence of variables in nonduplicate statements.
- Step 2: Determine the ordering constraints after local independence transformations.
- Step 3: Move the transformed statements out of the clone if the motion preserves the semantics.

We provide more detail about each step of the framework in the following sections using a code clone shown in Figure

```
          ......
 ++(1)    if( a < b ){
 ++(2)      k = a;
   (3)      n = m + b;
 ++(4)    } else {
 ++(5)      k = b;
 ++(6)    }
 ++(7)    d = m;
   (8)    j = d + 1;
 ++(9)    while( k < j) {
++(10)      k = k + 1;
++(11)      n = n + k;
++(12)    }
  (13)    d = j + 1;
++(14)    j = a + n;
          ......
```

Fig. 4
A CODE CLONE

```
          ......
 ++(1)    if( a < b ){
 ++(2)      k = a;
   (3)      n = m + b;
 ++(4)    } else {
 ++(5)      k = b;
 ++(6)    }
 ++(7)    d = m;
  (8')    j = m + 1;
 ++(9)    while( k < j) {
++(10)      k = k + 1;
++(11)      n = n + k;
++(12)    }
 (13')    d = m + 1 + 1;
++(14)    j = a + n ;
          ......
```

Fig. 5
THE CODE CLONE AFTER LOCAL INDEPENDENCE TRANSFORMATIONS

4, assuming that lines 1, 2, 4, 5, 6, 7, 9, 10, 11, 12, and 14 are marked as duplicate when compared with the other similar clones in the program.

4.1 Determining Local Independence

In this step, we determine the local independence properties of variables in the code clone, and check for the possible local independence transformations.

For the code clone in Figure 4, the locally independent variables and the possible local independence transformations are listed as follows.

- *Locally independent variables*: Two locally independent variables are found: b and m.
- *Local independence transformations*: There are two possible local independence transformations.
  - For line (8), if we replace the variable d with m, the transformed statement is backward locally independent.
  - For line (13), if we replace the variable j with $m+1$, the transformed statement is forward locally independent.

After local independence transformations, the code clone is transformed into Figure 5. Note that line (13) can be further transformed into $d = m+2$ using constant folding [22], [23], [24].

4.2 Determining Ordering Constraints

In this step, we determine the ordering constraints for the code clone after local dependence transformations. For simplicity, we use the ordering constraints proposed by Komondoor and Horwitz [19].

For the code clone in Figure 5, the ordering constraints are determined as follows.

- *Data-dependence constraints*: $(2) \leq (9)$, $(5) \leq (9)$, $(8') \leq (9)$, $(2) \leq (10)$, $(5) \leq (10)$, $(2) \leq (11)$, $(3) \leq (11)$, $(5) \leq (11)$, and $(11) \leq (14)$.
- *Loop-structure constraint*: $(10) = (11)$.
- *Control-dependence constraints*: $(2) \Rightarrow (1)$, $(3) \Rightarrow (1)$, $(5) \Rightarrow (1)$, $(10) \Rightarrow (9)$, and $(11) \Rightarrow (9)$.
- *Extended constraints*: $(3) \leq (10)$, $(1) \leq (9)$, $(1) \leq (10)$, $(1) \leq (11)$, and $(3) \leq (9)$.

Based on the ordering constraints, we note the following results.

1) Moving line (8') to the beginning of the clone preserves the semantics, and
2) Moving line (13') to the end of the clone preserves the semantics.

4.3 Moving Nonduplicate Statements

In this step, we move the nonduplicate statements out of the clone if the motion preserves the semantics, according to the ordering constraints computed in Step 2.

For the code clone in Figure 5, two code motion transformations are performed:

- Line (8') is moved to the beginning of the clone, and
- Line (13') is moved to the end of the clone.

Figure 6 shows the code clone after moving the nonduplicate statements out of the clone. Note that now the clone has only 12 lines, with the lines (8') and (13') excluded.

5. Correctness

In this section, we analyze the correctness of our framework for removing nonduplicate statements in code clone using the techniques of local independence transformation and code motion.

The correctness proof of our framework is equivalent to the following theorem.

```
          ......
  (8')    j = m + 1;
 ++(1)    if( a < b ){
 ++(2)      k = a;
   (3)      n = m + b;
 ++(4)    } else {
 ++(5)      k = b;
 ++(6)    }
 ++(7)    d = m;
 ++(9)    while( k < j) {
++(10)      k = k + 1;
++(11)      n = n + k;
++(12)    }
++(14)    j = a + n ;
 (13')    d = m + 1 + 1;
          ......
```

Fig. 6
THE CODE CLONE AFTER MOVING NONDUPLICATE STATEMENTS

Key Theorem: Given a CFG $\mathcal{P}$ of a code clone $\mathcal{M}$, it is possible to create a new semantically equivalent CFG $\mathcal{P}_t$ with the same nodes as $\mathcal{P}$, except for a node q replaced by $q' = \mathcal{T}^r_x(q)$, where r is the uniquely immediate dominant of q in $\mathcal{P}$ with the definition of X. Let $\mathcal{P}_m$ be a permutation of P_t by moving q' either forward or backward with the ordering constraints satisfied. Then, $\mathcal{P}$ and $\mathcal{P}_m$ are semantically equivalent. □

The Key Theorem is actually composed of two parts:

1) Given a CFG $\mathcal{P}$ of a code clone $\mathcal{M}$, it is possible to create a new semantically equivalent CFG $\mathcal{P}_t$ with the same nodes as P, except for a node q replaced by $q' = \mathcal{T}^r_x(q)$, where r is the unique and nearest dominant of q in $\mathcal{P}$. Then, $\mathcal{P}$ and $\mathcal{P}_t$ are semantically equivalent.
2) Let $\mathcal{P}_m$ be a permutation of $\mathcal{P}_t$ by moving q' either forward or backward with the ordering constraints satisfied. Then, $\mathcal{P}_t$ and $\mathcal{P}_m$ are semantically equivalent.

We officially describe these two parts in the following lemmas.

Lemma 1: Given

1) a control flow graph $\mathcal{P}$ of a code clone $\mathcal{M}$,
2) a pair of nodes r, $q \in \mathcal{P}$, where r is the uniquely immediate dominant of q in $\mathcal{P}$ with the definition of x, and
3) $\mathcal{P}_t$, a new CFG with the same nodes as $\mathcal{P}$ except for q replaced by $q' = \mathcal{T}^r_x(q)$,

then $\mathcal{P}$ and $\mathcal{P}_t$ are semantically equivalent.

Proof Outline: The proof depends on the following properties.

- After execution of r, the value of x is the definition given at r, denoted as x_r.
- Since r is the uniquely immediate dominant of q in $\mathcal{P}$ with the definition of x, the value of x at q is exactly x_r.
- Since $q' = \mathcal{T}^r_x(q)$, the value of q' is eqivalent to q.

Hence, $\mathcal{P}$ and $\mathcal{P}_t$ are semantically equivalent. □

Lemma 2: Given

1) a control flow graph $\mathcal{P}$ of a code clone $\mathcal{M}$, and
2) $\mathcal{P}_m$, a permutation of $\mathcal{P}$ with the ordering constraints satisfied,

then $\mathcal{P}$ and $\mathcal{P}_m$ are semantically equivalent.

The proof is provided by Komondoor and Horwitz in [19]. □

This completes the proof that our framework for removing nonduplicate statements in code clones is correct.

6. Experiments

This section describes the implementation of the proposed framework and presents our experiments on the effectiveness of the framework.

We implement the proposed framework for removing nonduplicate statements in code clones in the C programming language and integrate into the gcc compiler, version 4.1.2. All the experiments are performed at a desktop computing system with an Intel 2.4GHz Core 2 Duo CPU running the Fedora Core 8 Linux operating system with 4 Giga-Byte memory. For a comparison, we also implemented the procedure extraction algorithm proposed by Komondoor and Horwitz [19].

Since our framework assumes that the code clones in the program are already detected with duplicate statements marked and nonduplicate statements unmarked, we modify the clone detector that we developed in [25] and [17] for this purpose. We set up a similarity threshold of 50%; that is, the number of allowed nonduplicate statements is up to 50% of the total number of statements in the code clone.

For the experiments, we apply the implemented framework to the source code of Wget system, version 1.10.1. The source code of Wget system includes 33 .c files and 29,771 statements in total.

The measured statistics are listed as follows.

- There are 3,382 statements in code clones, which is 11.36% of the all statements.
- There are 534 nonduplicate statements in code clones, which is 17.27% of all the statements in clones.
- The algorithm by Komondoor and Horwitz moves 387 nonduplicate statements out of the clones, which is 72.47% of all nonduplicate statements in the clones.
- Our framework moves 416 nonduplicate statements out of the clones, which is 77.9% of all nonduplicate statements in the clones.

As a summary, in the experiments on the Wget source code, our framework successfully removes 77.9% of the nonduplicate statements in the code clones. To compare with the algorithm by Komondoor and Horwitz, our framework removes 29 more nonduplicate statements, which is 5.43% more.

7. Conclusion

In this paper, we describe a new program transformation, called local independence transformation, and its application to removing nonduplicate statements in the code clones. The local independence transformation analyzes the local independence properties of variables in the program and transforms the locally dependent statements into backward or forward locally independent statements when possible.

We propose a three-step framework for removing nonduplicate statements in the code clones using the techniques of local independence transformation and code motion. When extracting duplicate statements into procedures, the nonduplicate statements in the clones induce a considerable overhead. According the analysis of local independence property of variables, our framework transforms the locally dependent statements into backward or forward locally independent statements when possible. The transformed backward or forward locally independent statements may be moved out of the clones with the semantics of the program preserved.

We implement the proposed framework and integrate into the gcc compiler. In the experiment on the source code of Wget system, our framework moves 77.9% of the nonduplicate statements out of the clones, which is an improvement of 5.43% over the Komondoor and Horwitz's approach in terms of the number of nonduplicate statements moved out of the clones.

Acknowledgement

This work is partially supported by the National Science Council under grants NSC97-2218-E-259-001, NSC98-2220-E-259-002, and NSC99-2220-E-259-001.

References

[1] I. Semantic Designs, *Clone Doctor: Software Clone Detection and Removal*. http://www.semdesigns.com/: Available at the web site of Semantic Designs, Inc., 2010.

[2] B. S. Baker, "A program for identifying duplicated code," in *Computing Science and Statistics Proceedings of the 24th Symposium on the Interface*. College Station, Texas: Interface Foundation of North America, Mar. 1992, pp. 49–57.

[3] ——, "On finding duplication and near-duplication in large software systems," in *Proceedings of the Second Working Conference on Reverse Engineering*. Washington, DC: IEEE Computer Society, Jul. 1995, pp. 86–95.

[4] ——, "Finding clones with dup: Analysis of an experiment," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 608–621, Sep. 2007.

[5] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Proceedings of the International Conference on Software Maintenance (ICSM'98)*. Washington, DC: IEEE Computer Society, Nov. 1998, pp. 368–377.

[6] C. Yung, Y.-C. Lai, and Q.-X. Tu, "Refining large clones using local independence transformation and code motion," Lab. of Compiler Technology and Application, CSIE Dept., National Dong Hwa University, Tech. Rep. CTA-TR-201101, Jan. 2011.

[7] S. Ducasse, M. Rieger, and S. Demeyer, "A language independent approach for detecting duplicated code," in *Proceedings of the IEEE International Conference on Software Maintenance (ICSM '99)*. Washington, DC: IEEE Computer Society, Aug. 1999, pp. 109–118.

[8] M. Harman, D. Binkley, R. Singh, and R. M. Hierons, "Amorphous procedure extraction," in *Proceedings of the Fourth IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 04)*. Washington, DC: IEEE Computer Society, Apr. 2004, pp. 85–94.

[9] J. H. Johnson, "Substring matching for clone detection and change tracking," in *Proceedings of International Conference on Software Maintenance (ICSM '94)*. Washington, DC: IEEE Computer Society, Sep. 1994, pp. 120–126.

[10] T. Kamiya, S. Kusumoto, and K. Inoue., "Ccfinder: a multi-linguistic token-based code clone detection system for large scale source code," *IEEE Transactions on Software Engineering*, vol. 28, no. 7, pp. 654–670, Jul. 2002.

[11] J. Krinke, "Identifying similar code with program dependence graphs," in *WCRE '01: Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*. Washington, DC: IEEE Computer Society, Oct. 2001, pp. 301–309.

[12] F. Lanubile and G. Visaggio, "Extracting reusable functions by flow graph-based program slicing," *IEEE Transactions on Software Engineering*, vol. 23, no. 4, pp. 246–259, Apr. 1997.

[13] C. Yung and S.-L. Chen, "High-level procedural abstraction for reducing size of object code," in *Proceedings on the International Computer Symposium 2002*, vol. 1. Hualien, Taiwan: ICS 2002, Dec. 2002, pp. 165–172.

[14] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 577–591, Sep. 2007.

[15] F. V. Rysselberghe and S. Demeyer, "Evaluating clone detection techniques from a refactoring perspective," in *Proceedings of the 19th IEEE International Conference on Automated Software Engineering (ASE '04)*. Washington, DC, USA: IEEE Computer Society, Sep. 2004, pp. 336–339.

[16] J. Mayrand, C. Leblanc, and E. Merlo, "Experiment on the automatic detection of function clones in a software system using metrics," in *Proceedings of the 1996 International Conference on Software Maintenance (ICSM '96)*. Washington, DC: IEEE Computer Society, Nov. 1996, pp. 244–253.

[17] C. Yung and C.-W. Wu, "A new approach to parameterized clone detection using abstract syntax tree," in *Proceedings on the* 14^{th} *Workshop on Compiler Techniques for High Performance Computing*. Taipei, Taiwan: Session I, CTHPC 2008, May 2008, pp. 11–20.

[18] R. Komondoor and S. Horwitz., "Eliminating duplication in source code via procedure extraction," Dept. of Computer Sciences, Univ. of Wisconsin-Madison, Tech. Rep. 1461, 2002.

[19] ——, "Semantics-preserving procedure extraction," in *Proceedings of the 27th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '00)*. New York: ACM, Jan. 2000, pp. 155–169.

[20] R. Komondoor and S. Horwitz, "Effective, automatic procedure extraction," in *IWPC '03: Proceedings of the 11th IEEE International Workshop on Program Comprehension*. Washington, DC: IEEE Computer Society, May 2003, pp. 33–42.

[21] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Transactions on Programming Languages and Systems*, vol. 9, no. 3, pp. 319–349, Jul. 1987.

[22] D. Grove and L. Torczon, "Interprocedural constant propagation: a study of jump function implementation," in *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation (PLDI '93)*. New York: ACM, Jun. 1993, pp. 90–99.

[23] R. Metzger and S. Stroud, "Interprocedural constant propagation: an empirical study," *ACM Letters on Programming Languages and Systems*, vol. 2, no. 1-4, pp. 213–232, Mar. 1993.

[24] C. Yung and H.-H. Wang, "Constant propagation for loops with factored use-def chains," in *Proceedings on the* 8^{th} *Workshop on Compiler Techniques for High Performance Computing*. Hualien, Taiwan: CTHPC 2002, Mar. 2002, pp. 95–101.

[25] C. Yung and Y.-H. Fang, "An empirical analysis of program size reduction by parameterized procedure abstraction," in *Proceedings of the 10th International Conference on High-Performance Computing in Asia-Pacific Region*. Kaohsiung, Taiwan: HPC Asia 2009, Mar. 2009, pp. 424–431.

Patterns-Based Assistance for Temporal Requirement Specification

Ahmed MEKKI[1,2], **Mohamed GHAZEL**[1,2] **and Armand TOGUYENI**[1,3]
[1]Univ Lille Nord de France F-59000 Lille, France
[2]IFSTTAR, ESTAS, F-59666, Villeneuve d'Ascq, France
[3]EC LILLE, LAGIS, F-59651, Villeneuve d'Ascq, France
{ahmed.mekki, mohamed.ghazel}@ifsttar.fr, armand.toguyeni@ec-lille.fr

Abstract—***R**equirement **S**pecification (**RS**) presents a baseline for the validation/verification step. Therefore, errors within the specification phase involve huge financial burdens (release delay, system cost raise). In fact, the RS step is tedious and an error-prone since the user usually have to handle abstract notation as well as mathematical-based languages within this step. To deal with this issue, we present a pattern-based method for assisting the user during the RS. Indeed, this method predates by defining a new typology taking into account all the common temporal requirements one may meet when dealing with critical systems. Then, in order to provide the user with specification means which are at the same time simple, intuitive and rigorous, we have developed a literal word-based formal grammar able to express all the types of the identified requirements. Furthermore, a generic repository of observation patterns relative to the new time constraint taxonomy is proposed. Concretely, to check the temporal aspects of a given specification, the observation patterns relative to the identified and extracted requirements are instantiated to obtain appropriate observers which will play the role of watch-dog for the system to check.*

Keywords: Temporal Requirements, Natural-Language, System Specification, Observation Patterns

1. Introduction

Since it presents a baseline for validation and verification, **R**equirement **S**pecification (**RS**) is one of the most capital research topics in critical systems engineering. Consequently, its implication on the safety and the correctness of critical systems (e.g. nuclear plants, medical devices, transportation systems) is generally incontestable, since such systems must achieve a high level of robustness and reliability. On the other hand, critical systems usually involve time-dependent functionality, and therefore, tools and techniques for behavior design and verification (especially temporal requirements) are increasingly important and strongly recommended. The techniques most used are Temporal Logic and **T**imed **A**utomata (**TA**) [3]. The former is generally used for expressing requirements, while the later is used for specifying timed systems. Both of them, temporal logic and TA, are well suited for expressing timed behavior and for modeling real-time components. This is proven by the number of developed automatic verification tools (model-checkers) (e.g., Uppaal [13] and Kronos [21]) that use both of these methods. Moreover, theses tools have proven to be efficient. Nevertheless, specifying time-constraint properties is becoming a more and more difficult task, due to the widespread applications and increasing complexity of checked systems. Furthermore, this process (manipulating abstract operators to express requirement) is tedious and error-prone, since it requires sophisticated temporal logic knowledge, as well as mathematical skills.

The aim of the work presented here is to guide the user during the temporal requirement specification phase. Indeed, the cost of errors due to requirement specification identified during the system verification and validation is very important. For instance, a study (Figure 1)[12] presented by many different companies and organizations over the years shows the relative cost of requirement error correction over the software life-cycle. Indeed, the fix cost of an error identified during the requirement definition phase is about $1. However, the error fix cost increase rapidly as one proceeds in the system/software life-cycle. For instance, in the case where it is not identified until the operations step, an error cost from $40 to $1,000 to fix it. Errors are generally due to ambiguity, inaccuracy or inconsistency during the specification. To deal with this, Gerrit Muller proposed in [17] the criteria of a right RS:

- it should reflect the real needs of all stakeholders in a simple, implicit and latent way;
- it should describe a feasible product;
- it must answer most critical design questions;
- it should be useful for human product creators.

Consequently, assisting the user with some guiding means while specifying the temporal requirements will considerably reduce the number of errors and, therefore, helps the user to reduce and to manage the system cost.

As mentioned previously in this paper, we focus on the specification step of temporal requirements. In fact, the idea is to propose assisting means that are easy to manipulate and, at the same time, accurate and formal. Nevertheless, the more accurate and rigorous a notation/language is, the more abstract and difficult to handle and understand it becomes. Therefore, one of the challenges that we faced while dealing

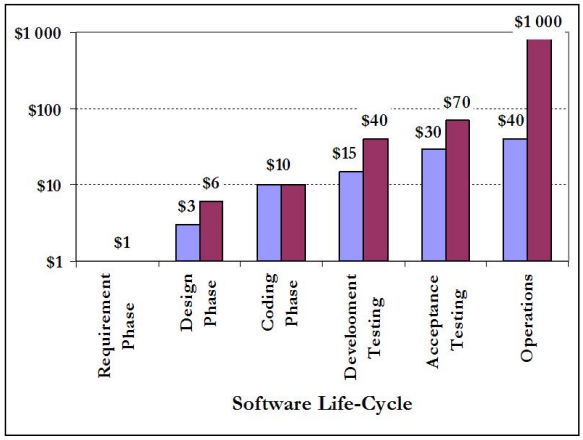


Fig. 1: Requirement Errors Fix Cost [12]

with this work was to look at both intuition/simplicity and rigor/accuracy. To deal with this, first, we propose a taxonomy that identifies all the common temporal constraints that one can meet within time-constrained systems. Then, based on this taxonomy, we developed a natural-language grammar (denoted **SEG** hereafter) able to cover all temporal property types. Indeed, **SEG** (**S**tructured **E**nglish **G**rammar) is a set of literal compoundable elements with a precise and rigorous interpretation. In this way, formal aspects -met when dealing with requirement specification- are hidden to the user, since he will only handle literal terms to make assertions in order to express requirements. Once introduced, the temporal requirements classification as well as the SEG will be associated to an observation pattern repository. Indeed, the idea to associate to each requirement type an observation pattern (observer) in order to check the validation/violation of the requirement. In other words, the observer will act as watch-dog to the associated requirement. Therefore, each generated sentence from SEG identifies a unique requirement type and, consequently, identifies a unique observer and vice-versa.

The paper is organized as follows: in Section 2, an overview of the context and related works is given. In Section 3, we present the classification of temporal requirements we have established. The developed formal grammar for specifying temporal requirements is introduced in Section 4. Section 5 draws an overview of a validation method that we are developing, before concluding and suggesting some future works in Section 6.

2. Context and Related Work

Several studies dealing with the analysis and classification of temporal requirements have been proposed and published. The most referenced classifications are Allen [1], Dwyer [7], Konrad [10] and Sadani [18].

- The Allen classification [1] is the classification most referenced in the artificial intelligence field. He proposes a process-based taxonomy where he defines 14 possible relations between processes, such as **meets**, **overlaps**, **starts**, **finishes**, etc. Nevertheless, event-based requirements, as well as quantitative properties, could not be expressed using this classification.
- The Dwyer classification [7] is a taxonomy of time-dependent constraints. Dwyer formally expressed and defined all the identified constraint using CTL, LTL, QRE and GIL temporal logic formulas. Furthermore, Dwyer introduced "**scope**" in the requirement expression. **Scope** is used to express the applicability interval of the constraints. Dwyer introduced five scopes: **Globally**, **Before an event/state occurs**, **After an event/state occurs**, **Between two events/states** and, finally, **Until an event/state**. Nevertheless, despite the fact that this classification deals with state and events, it only deals with qualitative temporal requirements. In fact, one cannot express either quantitative constraints or punctuality property using the Dwyer classification.
- The Konrad classification [10] is a real-time extension to the Dwyer classification. This classification uses MTL, TCTL, and RTGIL temporal logics to express formally quantitative requirements. Like Dwyer, the Konrad classification proposes the use of the five scopes mentioned above. Nevertheless, his classification still suffers from the absence of the punctuality constraint, as well as, the absence of interval requirements.
- The Sadani classification [18] is an interval-based taxonomy. This classification is very suitable for expressing requirements on process. Nevertheless, despite the fact that the punctuality requirement is considered, requirements on events cannot be specified using this classification, since it is interval-logic-based. Furthermore, no formal notation is used to define requirements in a formal way.

3. Temporal Requirement Taxonomy

Here we propose a new classification which is, in some way, completely different compared to those mentioned previously in Section 2. The first difference is in the way requirements are organized and the second is the numerous types of requirement which we can express and which the others cannot. In fact, requirements are split into response, precedence and absolute presence subsets. Moreover, our classification refers only to events (factual or fictive), which improves the coherence and the clarity of requirements. In practice, for requirements dealing with states/processes, instead of directly considering these states/processes, we express the requirements using two events: the first event represents the activation of the state/process and the second the deactivation. In this way, we obtain a repository of event-based requirements, as depicted in the shape of a UML class-diagram of Figure 2. The definition of each requirement type is given in Table 1.

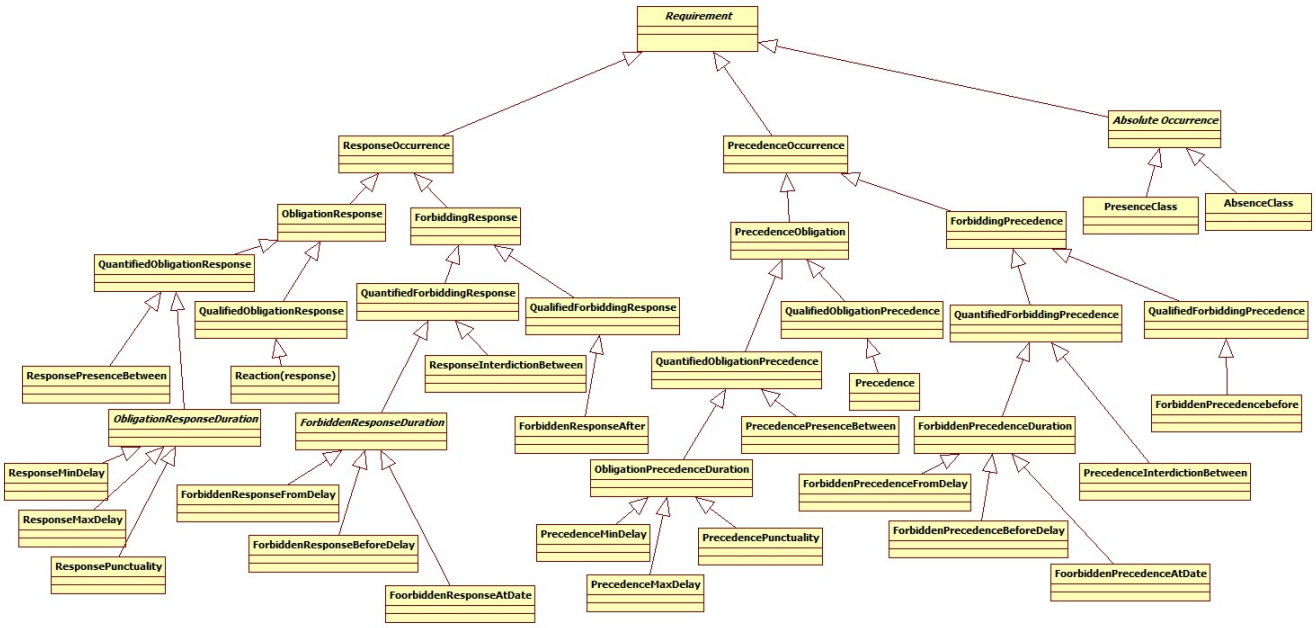


Fig. 2: Temporal Requirements Classification

As shown in the Dwyer classification, requirements could be monitored permanently or restricted to a special context. The scope of a given requirement defines the applicability interval in which it has to be applied. Here we consider 4 scopes, namely:

- **Globally**: the requirement must be satisfied all the time and this is the scope by default,
- **After a Reference**: the requirement must be satisfied all the time after a given reference,
- **Before a Reference**: the requirement must be satisfied all the time before a given reference,
- **Between two References**: the requirement must be satisfied all the time after a given begin-reference and before an end-reference.

4. Structured English Grammar

4.1 Idea

Nowadays, critical systems are usually widespread and put together components that come from different manufactures. Consequently, this increase rapidly the complexity of specification, as well as, the complexity of verification/validation of such systems. In other words, RS of critical systems is becoming challenging for system designers, as well as, for automatic verification tool developers. Furthermore, despite the higher level of automation of verification tools (**Model-checking** [4]: Uppaal, Kronos, Spin, ...; **theorem-proven**: HOL, Isabelle, LP, PVS, ...), users often have to write logic formula to express system properties to be checked. Several Temporal Logics (**LTL** [14], **MTL** [11], **CTL** [5] and **TCTL** [2]) have been suggested in order to specify timing constraints. Although using a formal notation allows a rigorous specification that is well supported by tools this remains tedious and error-prone, since formal notations require sophisticated logical and/or mathematical skills. Thus, the idea is to assist the user with some guiding means which are intuitive and simple, and at the same time, precise, rigorous and expressive.

The idea is to propose a supporting approach that hides the formal foundation from the user. In fact, we propose a formal specification grammar that covers all the common requirements one may meet within critical systems. Then, based on this formal grammar, we will derive some compoundable literal-based elements which will be handled by the user. In addition, we will embed the grammar generation rules within the composition procedure. Consequently, the user will be guided in such a way to obtain precise assertions while handling simple literal-based constructs.

4.2 BNF notation

4.2.1 Definition

A Formal Grammar is a formalism allowing the definition of a precise syntax. In fact, a formal grammar determines the set, finite or infinite, of admissible strings on a given alphabet, i.e the strings are combinations of alphabet elements which are acceptable in a precise domain. The notation most used for defining formal grammar is **B**ackus-**N**aur **F**orm (**BNF**) [19]. BNF is a formal notation that is usually used to describe formal languages. It provides a concise and accurate method for describing possible modes of combination of

Table 1: Temporal Requirement's Taxonomy Descriptions

	Class	Sub Class	Category	Sub Category	Pattern Name	Description
Requirements	Response	Obligation Response	Quantified Obligation Response		**Response Presence Between**	**R** ensures that an event (E_{mon}) must occur within a temporal interval [t_{Begin}; t_{End}] from "i" occurrence of event E_{Ref}
				Obligation Response Duration	**Response Min Delay**	**R** ensures that an event (E_{mon}) must occur after a minimum delay T_{min} time unit from "i" occurrence of event E_{Ref}
					Response Max Delay	**R** ensures that an event (E_{mon}) must occur before a maximum delay T_{max} time unit from "i" occurrence of event E_{Ref}
					Response Punctuality Delay	**R** ensures that an event (E_{mon}) must occur exactly at a delay T time unit from "i" occurrence of event E_{Ref}
			Qualified Obligation Response		**Reaction**	**R** ensures that an event (E_{mon}) must occur after "i" occurrence of event E_{Ref}
		Forbidding Response	Quantified Forbidding Response		**Response Interdiction Between**	**R** ensures that an event (E_{mon}) cannot occur within a temporal interval [t_{Begin}; t_{End}] from "i" occurrence of event E_{Ref}
				Forbidden Response Duration	**Forbidden Response From a Delay**	**R** ensures that an event (E_{mon}) cannot occur after a minimum delay T_{min} time unit from "i" occurrence of event E_{Ref}
					Forbidden Response Before a Delay	**R** ensures that an event (E_{mon}) cannot occur before a maximum delay T_{max} time unit from "i" occurrence of event E_{Ref}
					Forbidden Response at Date	**R** ensures that an event (E_{mon}) cannot occur exactly at a delay T time unit from "i" occurrence of event E_{Ref}
			Qualified Interdiction Response		**Forbidden Response**	**R** ensures that an event (E_{mon}) cannot occur after "i" occurrence of event E_{Ref}
	Precedence	Precedence Obligation	Quantified Obligation Precedence		**Precedence Presence Between**	**R** ensures that an event (E_{mon}) must occur within a temporal interval [t_{Begin}; t_{End}] before "i" occurrence of event E_{Ref}
				Obligation Precedence Duration	**Precedence Min Delay**	**R** ensures that an event (E_{mon}) must occur after a minimum delay T_{min} time unit before "i" occurrence of event E_{Ref}
					Precedence Max Delay	**R** ensures that an event (E_{mon}) must occur before a maximum delay T_{max} time unit before "i" occurrence of event E_{Ref}
					Precedence Punctuality Delay	**R** ensures that an event (E_{mon}) must occur exactly at a delay T time unit before "i" occurrence of event E_{Ref}
			Qualified Obligation Precedence		**Precedence**	**R** ensures that an event (E_{mon}) must occur before "i" occurrence of event E_{Ref}
		Forbidding Precedence	Quantified Forbidding Precedence		**Precedence Interdiction Between**	**R** ensures that an event (E_{mon}) cannot occur within a temporal interval [t_{Begin}; t_{End}] before "i" occurrence of event E_{Ref}
				Forbidden Precedence Duration	**Forbidden Precedence From a Delay**	**R** ensures that an event (E_{mon}) cannot occur after a minimum delay T_{min} time unit before "i" occurrence of event E_{Ref}
					Forbidden Precedence Before a Delay	**R** ensures that an event (E_{mon}) cannot occur before a maximum delay T_{max} time unit before "i" occurrence of event E_{Ref}
					Forbidden Precedence at Date	**R** ensures that an event (E_{mon}) cannot occur exactly at a delay T time unit before "i" occurrence of event E_{Ref}
			Qualified Interdiction Precedence		**Forbidden Precedence**	**R** ensures that an event (E_{mon}) cannot occur before "i" occurrence of event E_{Ref}
	Absolute Occurrence				**Absence**	**R** ensures that an event (E_{mon}) cannot occur
					Presence	**R** ensures that an event (E_{mon}) must occur

constituents. In other words, it shows exactly how to build the recursive data structures for syntax trees during the language implementation process.

4.2.2 BNF Foundation

Developed by John Backus and Peter Naur, BNF is a metasyntax widely used to express context-free grammars. In fact, it is used to define the syntax of a programming language by using a set of rules (sometimes called productions). Each rule describes one possible way of constructing a constituent belonging to a syntactic category (non-terminal elements). In practice, we write the names of syntactic categories at the beginning of the rule within angle brackets (e.g.: <expression>). On the other hand, the right side of the rule lists the components of the constituents. These components could be syntactic elements (terminal elements), other syntactic categories (non- terminal elements), or both. Moreover, the rule right side components are listed in order and without limit on the number of components. At the beginning of the rule, the non-terminal elements are

separated from the right-hand side by the symbol **::=**. Several rules could be combined into one by concatenating their right-hand sides *if and only if* they have the same non-terminal name. The combination is made using vertical bars to separate alternative constructions (e.g.,

- <**R**>::=<A> and
- <**R**>::=<B>
- ⇒ <**R**>::=<A>|<B>)

The vertical bar (|), angle brackets (<,>) and the ::= symbol are parts of the meta-notation and not parts of the programming language that is being described. Hereafter, the BNF notation is expressed using BNF itself:

<syntax>	::= {<rule>};
<rule>	::= <identifier> "::=" <expression>;
<expression>	::= <term> {"|" <term>};
<term>	::= <factor> {<factor>};
<factor>	::= <identifier>| <quotedSymbol> | "("<expression>")" | "["<expression>"]" | "{"<expression>"}";
<identifier>	::= letter {letter | digit};
<quotedSymbol>	::= "'" anyCharacter "'" | '"' anyCharacter '"';

4.3 Developed SEG

To facilitate the expression and the formalization of temporal properties, we propose a structured English grammar (**SEG**). This grammar supports both qualitative and quantitative properties. The aim of developing such a grammar is to provide the user with a simple means for expressing temporal requirements. The SEG is a guiding framework which offers a means of expression to the user. It also constrains him to make assertions according to a predefined syntax, hence making it possible to automatically recognize the expressed requirements. Concretely, each sentence generated by our grammar describes one temporal property and serves as a handler that aids expressing and understanding the requirement. SEG is expressed below using BNF (Backus-Naur Form) notation:

<Property>	::= <scope> ":" <specification>;
<scope>	::= "global" | "before" < entity > | "after" < entity> | "between"' <entity > "and"' <entity>;
<Specification>	::= <entity> <obligation> <occurrenceType>;
<entity>	::= <event> | activate(<state>) | deactivate(<state>);
<obligation>	::= "must" | "cannot";
<occurrenceType>	::= ("occur" [<referenceOccur>])|("precede" <referencePrecede>);
<referenceOccur>	::= ([<punctuality > | <fromAdelay> | <BeforeAdelay> | <Between>] "after") (<Repetition> | <LastOcc>);
<referencePrecede>	::= ([<punctuality > | <fromAdelay> | <BeforeAdelay> | <Between>] "before") <OccurrenceOfRef>;
<punctuality>	::= "exactly" <delay>;
<fromAdelay>	::= "after a minimum delay of" <delay>;
<BeforeAdelay>	::= "before a maximum delay of" <delay>;
<Between>	::= "between a minimum delay of" <delay> "and a maximum delay of" <delay>;
<Repetition>	::= <OccurrenceOfRef> and (("Regardless following occurrences") | ("before the next occurrence"));
<OccurrenceOfRef>	::= "i th occurrence of" <entity>;
<LastOcc>	::= "the last occurrence of" <entity>;
<delay>	::= <integer> "timeunit";
<integer>	::= <digit>+;
<digit>	::= {0|1|2|3|4|5|6|7|8|9};
<i>	::= <number> | <digit> <i> | <i> <digit>;
<number>	::= {1|2|3|4|5|6|7|8|9};

An assertion obtained using the specification above is a composition of:

- a scope: indicates the applicability interval (range) on which the requirement should be checked,
- an entity: represents the monitored entity, which could be an event, a state/process activation or a state/process deactivation,
- an obligation: expresses the obligation (must occur) or the prohibition (cannot occur). The analysis of the various case studies shows that all the temporal requirements can be expressed while combining must/cannot elementary assertions,
- an occurrence type: indicates the type of occurrence. Three types are used in our case; Response, Precedence and Absolute Occurrence.

Comparing our work to the works previously mentioned, we note:

- As mentioned above, here we present a classification and grammar that deals only with events. In fact, unlike Dwyer and Konrad, and in order to ensure homogeneity between requirement declarations, the requirements dealing with states/processes are brought back to deal with events while considering 2 events: the activation and deactivation of the involved states/processes,
- Repetitive events are not considered in previous works. Here, in this study, we give the user the possibility to express requirements that refer not only to an event, but also to the occurrence number "i" of this event,
- More requirement types can be expressed using our work. For instance, one can express interval constraints using new elementary requirements that we defined and which cannot be expressed using the previously mentioned works.

5. Verification Method

Both, temporal requirement classification and SEG, are used as a start step for a new verification method of temporal requirements. In this section, first, we introduce the next step of this new verification method which is the observation pattern repository. Then, we outline briefly an overview of the global method that we are developing.

5.1 Observation Pattern Basis

A pattern is a commonly reusable model in software systems that guarantees a set of characteristics and functionalities. The identification of a pattern is based on the context in which it is used. The goal behind developing patterns is to offer a support for system design and development [6]. Moreover, using patterns helps in keeping design standardized and useful, and minimizes the reinventing in the design process, since they facilitate reusability and knowledge capitalization [8].

In this work, we have set up a repository comprising generic timed automata observation patterns to watch all

the common temporal requirements that we defined in Table 1. An observer [9] is a TA model with, inter alia, a specific error-state ***"KO"*** reached as soon as the associate requirement is violated. The basis of patterns is introduced regardless of the systems' specification. Moreover, each generated sentence from our SEG identify a unique TA observation patterns and, in the same way, each pattern is associated to a unique generated sentence. In practice, all the observers made up for all the extracted requirements will be disposed in parallel to watch system behaviour.

Once defined, the pattern repository will be used in order to set up timed automata observers for the temporal requirements extracted from the specification of the system to be checked [16].

5.2 Verification process

In practice (Figure 3), the temporal requirements extracted (using our SEG) from the specification of the system to check are classified based on their types. For each extracted requirement from the investigated system, the suitable observation patterns (See Section 5.1) are picked up and instantiated. This instantiation step generates a set of TA Observers for the monitored requirements. On the other hand, the system to check model, which is depicted in the shape of Unified Modeling Language (UML) State Machines (SM) [20] models, is automatically translated into more formal notation, the TA, which provides support for analytic verification. The translation is done according to a transformation algorithm that we have developed [15]. The generated model is synchronized with the instantiated TA observers to obtain a global model. Hence, the verification task is performed on the basis of the obtained global model, with a reachability analysis while checking whether the observers forbidden states -corresponding to requirements violation- are reachable. A back-end tool, like Uppaal [13], is very suitable for carrying out such investigation.

6. Conclusions and Perspectives

In this paper we aim at introducing a means to assist and guide the user while specifying temporal requirements. First, we presented a new classification of all the common temporal requirements one may meet when dealing with specification of time-constrained systems. The advantages of our classification, unlike existing related works, are: (1) exclusively event-based, (2) ability to express more types of temporal requirements and, finally, (3) dealing with repetitive events. Second, we introduced a formal structured English grammar (SEG) for describing temporal requirements we have identified within the new classification. In fact, SEG consists of a set of natural-language/literal insertions with rigorous semantics. The aim of defining the SEG consists in hiding from the user formal aspects, as well as, abstract notations while specifying temporal requirements. Hence, SEG syntax is intuitive although, at the same time, rigorous

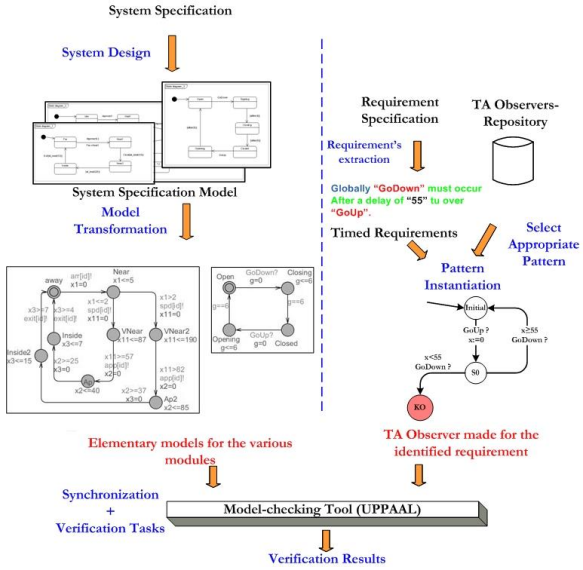


Fig. 3: Verification Process Architecture

and based on formal foundations. Third, we showed how this new classification, as well as the new SEG, are used in defining an observation pattern repository.

Once our observation patterns basis is implemented, we introduced a verification process based on this basis. Concretely, the method is composed of four processes: (1) First, from the requirements description, the temporal requirements are expressed by SEG assertions. Once expressed, the type of each requirement is determined, since, for each requirement, we have associated a generated sentence and vice-versa. (2) Second, for each requirement the appropriate TA observation pattern is picked up from the generic repository, then instantiated to produce an associate observer. This process results in some TA observers. Each TA observer watches an elementary temporal requirement. (3) In parallel, the specification of the system to be checked is abstracted and translated into TA models. (4) Finally, the instantiated TA observers are synchronized with the obtained (via model transformation) TA models to generate a global model holding both the specification of the system to be checked and the requirement monitoring. Consequently, the verification task is reduced to an error-state (***"KO"*** node) reachability analysis on the global model obtained.

Another contribution of our work consists in allowing the definition of more complex temporal requirements. In fact, experience showed us that one may need sometimes to define requirements that are a conjunction of requirements. Thereby, we defined patterns that handle various relations linking elementary requirements or even linking complex requirements. Indeed, relations over (elementary or complex) requirements such as ***"AND"***, ***"OR"*** and "***XOR***" can be defined using our patterns basis. To our knowledge, it is the first work that has proposed such observation patterns.

One following step consists in extending the current SEG to deal with requirements that refer to other requirements. Indeed, the idea is to allow the user to express requirements where the reference events as well as the monitored events refer to validation or violation of other requirements, and not only to events that come from the checked specification. Besides, this step seems easily implementable, since we presented an event-based classification as well as event-based pattern repository. However, a close view is still needed.

References

[1] J.F. Allen, *Towards a general theory of action and time*, Artificial Intelligence, 23: 123-154, 1984.

[2] R. Alur, *Techniques for automatic verification of real-time systems*, PhD thesis, Stanford University, 1991.

[3] R. Alur, and D. Dill, *A theory of timed automata*, Theoretical Computer Science, 126(2): 183-235, 1994.

[4] E.M. Clarke, *The Birth of Model Checking*, 25 Years of Model Checking: History, Achievements, Perspectives, Lecture Notes in Computer Science, Springer-Verlag, 1-26, 2008.

[5] E.M. Clarke, E.A. Emerson, and A.P. Sistla, *Automatic verification of finite-state concurrent systems using temporal logic specifications*, ACM Transactions on Programming Languages and Systems, 2:244-263, 1986.

[6] J. S. Dong, P. Hao, S. Qin, J. Sun, and W. Yi, *Timed automata patterns*. IEEE Transactions on Software Engineering, 34(6): 844Ű859, 2008.

[7] M.B. Dwyer, G.S. Avrunin, and J.C. Corbett, *Patterns in Property Specifications for Finite-State Verification*, In: 21st International Conference on Software Engineering, 411-420, IEEE Computer Society Press, 1999.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[9] M. Ghazel, A. Toguyeni, and P. Yim, *State observer for DES under partial observation with time petri nets*, Journal of Discrete Event Dynamic Systems, 19(2): 137-165, 2009.

[10] S. Konrad, and B.H.C. Cheng, *Real-time Specification Patterns*, In: 27th International Conference on Software Engineering (ICSE05), St Louis, MO, USA, 2005.

[11] R. Koymans, *Specifying real-time properties with metric temporal logic*, Real-Time Systems, 2(4): 255-299, 1990.

[12] W. Kuffel, *Extra Time Saves Money*, Computer Language, December 1990.

[13] K.G. Larsen, P. Pettersson, and W. Yi, *Uppaal in a nutshells*, International Journal of Software Tools for Technology Transfer, 1(1/2): 134-152, 1997.

[14] Z. Manna, and A. Pnueli, *The temporal logic of reactive and concurrent systems*, Springer-Verlag New York, 1992.

[15] A. Mekki, M. Ghazel and A. Toguyeni, *Time-constrained systems validation using MDA model transformation. A railway case study*, In Proceedings of the 8th ENIM IFAC International Conference of Modeling and Simulation, Hammamet, Tunisia, 2010.

[16] A. Mekki, M. Ghazel and A. Toguyeni, *Patterns for Temporal Requirements Engineering; A level crossing case study*, In Proceedings of the 7th International Conference on Informatics in Control, Automation and Robotics, Funchal, Madeira, Portugal, 2010.

[17] G. Muller, *What is a Good Requirement Specification?*, version: 0.2, (available on http://www.gaudisite.nl/), February 10, 2011.

[18] T. Sadani, P. De Saqui-Sannes, and J.P. Courtiat, *From RT-LOTOS to Time Petri Nets New Foundations for a Verification Platform*, SEFM, 250-260, 2005.

[19] Salmon, *Backus-naur forms*, Ed. Lavoisier, July 1992.

[20] *Unified Modeling Language Specification*, Version 2.2, OMG, 2009.

[21] S. Yovine, *Kronos: a verification tool for real-time systems*, International Journal of Software Tools for Technology Transfer, 1(1/2): 123-133, 1997.

SDP: Software Development Platform for Improving Software Productivity*

Zih-Jyun Song, Dyi-Rong Duh†, and Yi-Jung Chen
Department of Computer Science and Information Engineering
National Chi Nan University
Puli, Nantou Hsien 54561, Taiwan
{s98321516,drduh,yjchen}@ncnu.edu.tw

***Abstract* - In order to improve software productivity, this work designs and implements a Software Development Platform (SDP). The provided software development platform is composed of a reusable kernel program, a Graphical Input and Output Designer (GIOD) and a State Table Generator (STG). The kernel program is regarded as a Software Processor and driven by state tables each of which is transformed from a finite state diagram. The GIOD is used to visually design the graphical user interface (GUI); while the STG also provides a visual environment for designing finite state diagrams and can transform every designed finite state diagram into a finite state table including adopted I/O functions. Generally, the provided SDP can help designers to produce software more efficiently and be adapted to many kinds of programs.**

Keywords - software productivity; finite state diagram; software processor; software ICs; software development platform

I. INTRODUCTION

Software Engineers have devoted their efforts to improve software productivity for a very long time. However, software demands also increase substantially. In software development, coding and maintenance take a long time and cost a lot of money. Software engineers write one to two delivered lines of code per man-hour [13]. The relative cost for maintaining software and managing its evolution accounts for more than 90% of its total cost [8]. Therefore, to shorten the developing time and reduce the maintenance cost, it is important to have a tool to help software engineers develop and maintain their codes easily.

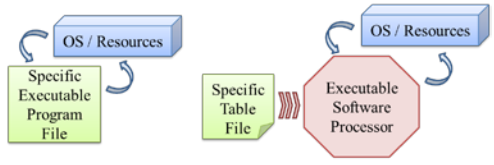
It is commonly known that information represented in graphics models and tabulations are easier for people to understand then information represented in text only. Therefore, software programs that have graphical descriptions are easier for engineers to perform coding and maintenances. One of the most commonly used graphical methods is the finite state machine (FSM) [7, 12], which is a mathematical abstraction and usually used to design digital logic or computer programs. FSM is also a popular model for describing software behavior. In 1987, Harel proposed statecharts [4], which includes the complexity relationship of states and the proprieties of transitions on the finite state machine. Additionally, Harel et al. in 1990 proposed STATEMATE [5] which is a working environment with code generators. STATEMATE are used to simulate the behavior of the state-charts [1]. Its code generator can transform finite state machines into C or JAVA codes. In 2000, the Unified Modeling Language (UML) [2, 11] was proposed by the Object Management Group (OMG), which is a standardized general purpose modeling language in software engineering. The UML defines a state machine diagram as its behavior diagram. It is more completely and concretely than statechart proposed by Harel. Gery et al. designed a development system called Rhapsody [3] in 2002, which has a code generator, too. The particular component of Rhapsody is an execution framework, which lets the designer simulate and debug at the modeling time.

However, the program generated by traditional design methods and code generator design methods, like STATEMATE and Rhapsody, are source program files specific to certain OS or architecture (as shown in Fig. 1(a)) which should be recompiled and linked to be a executable program. The troublesome of traditional method is that programmers usually write the same functions for many times just to adapt the function to different target platforms even with single instruction set architecture (ISA). If we can write the function once for all platforms with single ISA, the development cost can be significantly reduced.

In this paper, we propose a Software Development Platform (SDP), which provides tool for programmers to efficiently build a new program from existing functions. In SDP, the existing reusable executable programs (regarded as software ICs in this paper) in a library. Program designers represent the behaviors of specific algorithms by designing finite state diagrams in SDP. According to the finite state diagram specified by the designer, the result program is generated by SDP by concatenating the software ICs stored in the library. The proposed SDP tool also provides a Kernel Program, which is also regarded as the Software Processor in this paper. The Software Processor is driven by a finite state table that is transformed from a finite state diagram specified by the designer. As shown in Fig. 1(b), Software Processor reads the behavior of the program from the finite state table. If

* This work is partially supported by NSC 99-2221-E-260-010-.
† Correspondence to: D.-R. Duh; E-mail address: drduh@ncnu.edu.tw

the contents in the finite state diagram are the same as the ones in a traditional design method, the behavior of Software Processor will be the same as the executable program file generated by the traditional design method without recoding and recompiling. Notably, each generated state table may contain some input and output (I/O) functions which are included in software IC libraries. SDP provides a Graphical Input and Output Designer (GIOD) for designers to easily design a graphical user interface (GUI).



(a) (b)

Fig. 1 Outcome sketch charts of (a) the traditional design method and (b) SDP.

The rest of this paper is organized as follows. Section 2 gives an overview of SDP. Section 3 provides the SDP toolset and user interface. Section 4 demonstrates an example on SDP. Section 5 compares the traditional design approach with this work. Conclusions are finally drawn in Section 6.

II. Overview of the Platform

SDP is composed of the GIOD, STD and Software Processor. Fig. 2 illustrates the overview of the proposed SDP. Like designing hardware, the reusable Software Processor is a CPU; each generated state table is a program of the Software Processor, and every I/O function is regarded as a software IC. By the proposed GIOD, a GUI can be designed easily. The GIOD produces specific format called GIO object files. The STG provides an environment for visually building extended FSMs and transforming them into state tables accepted by the given Software Processor. The generated state tables may contain pointers of I/O functions contributed by GIOD.

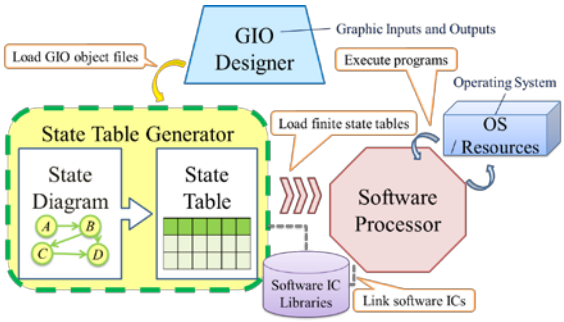
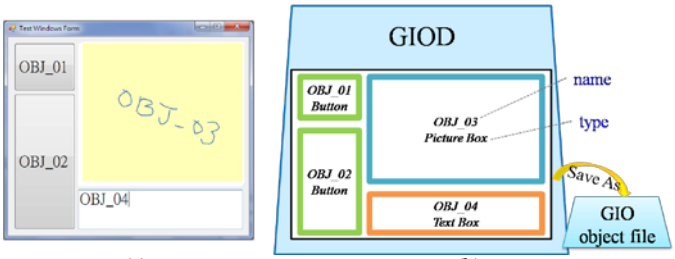


Fig. 2 Overview of SDP.

The continuing subsections introduce the detail of the GIOD, STG and Software Processor in turn.

A. GIOD

The GIOD is created for visually, diagrammatically and intuitionally designing the graphic user interfaces of Windows Forms. Windows Form Control objects provided by the GIOD include "Button" for clicking, "Text Box" for keying in some strings, "Picture Box" for displaying some pictures and so on. When a designer wants to have an object in his Windows Form application, the necessary information he needs is what the width and height of this object are, where this object on the form is, what the object's unique name is and what the Windows Form Control object's type is . Therefore, the arguments of each GIO object are size, location, name and type. For example, the designer can place a button object named "OBJ_01" with 100×200 pixels size in the upper left corner of the Windows From application. The designers choose desired objects provided by the GIOD to express their program's graphic interface. If a designer wants to design a Windows Form program with two buttons on the left side, one picture box and one text box on the right side as shown in Fig. 3(a), the designer should determine the names, sizes and locations of these objects and draw them as four GIO object blocks with the appropriate names, sizes and locations via the GIOD. Each GIO object's name can be keyed in from keyboard; every object's location can be changed by mouse dragging it to anywhere in the form, and each object's type is selected form one of the Windows Control objects' types on the GIOD. As shown in Fig. 3(b), all constructed GIO object blocks will be saved as a specific format called GIO object file in which each GIO object has its own size, location, name and type. The GIO object file can be read by STG for getting each GIO object's name as input and output objects, and read by the Software Processor for creating a real GUI of a Windows Form application with these Windows Form Control objects on it.



(a) (b)

Fig. 3 The concept of the GIOD. (a) A Windows Form program with four objects. (b) Using GIOD to design the desired objects.

B. STG and the Extended FSM

The STG can let designers draw their own finite state diagrams to describe the solutions for their requests. Then the STG transforms each finite state diagram into a finite state table.

Finite state diagram is one of the most conventional model for explaining software behavior. The regular final state machine (FSM) is formally denoted by the five-tuple $<Q, \Sigma, \delta, q_0, F>$ [9], where Q is a finite set of internal states; Σ is a finite and nonempty set of symbols, called input alphabet; δ is a total function, called transitional function; $q0 \in Q$ is an initial state, and $F \subseteq Q$ is the set of final states. The relationship between

transitional function δ and the internal states is written as (1). Equation (1) expresses that when the current state is q_{i-1} and it gets a symbol w_i, then the current state translates the state from q_{i-1} to q_i.

$$q_i = \delta(q_i-1, w_i), \text{ where } q_i \in Q \text{ and } w_i \in \Sigma \qquad (1)$$

In the traditional FSM, the input symbols cannot satisfy all the requisition of designing programs, including some input functions for expressing the users' actions more clearly and some input objects for driving input functions. Besides, the traditional FSM does not have the arguments to express outputs. Therefore, this work makes three modifications of the regular finite state diagram. First, the concept of the set of input symbol Σ is extended to a set of input function, named "*I*." Second, in order to show the specific transitional function, which is traversed, activate an output function before entering the next state. The set of output function is named "*O*". Third, in the cause of the input functions can be driven by any object, like keyboard or mouse, and the output functions can also drive any object, like screen or printer, so storing up a set of input objects and a set of output objects are named "*X*" and "*Y*," respectively. The transitional function δ is changed into (2).

$$(q_i, o_i, y_i) = \delta(q_{i-1}, i_i, x_i),$$
$$\text{where } q_i \in Q, i_i \in I, o_i \in O, x_i \in X \text{ and } y_i \in Y \qquad (2)$$

The extensional finite state machine is represented by a eight-tuple $<Q, I, O, X, Y, \delta, q_0, F>$, where the increases arguments, includes *I* is a set of input functions; *O* is a set of output functions; *X* is a set of input objects, and *Y* is a set of output objects. Fig. 4(a) shows the regular finite state diagram and Fig. 4(b) illustrates the extended finite state diagram. By the provided tool of this work which called STG, the extensional finite state diagram can be transformed into a finite state table automatically.

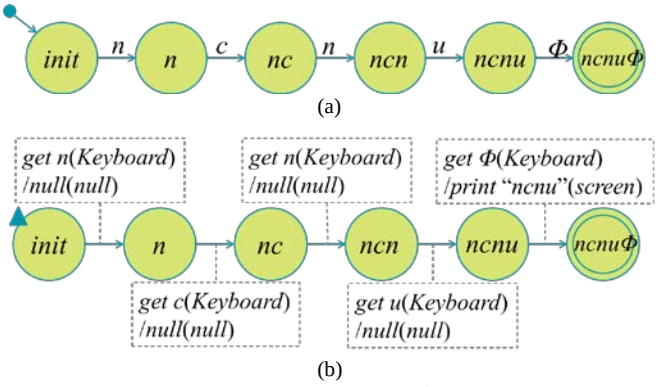


Fig. 4 Finite state diagrams. (a) The regular finite state diagram and (b) the extended finite state diagram.

As shown in Fig. 4(b), each transitional function δ is denoted as an arrow, which points at the target state q_i from the current state q_{i-1}. The input and output can be expressed by "*input function*(*input object*)/*output function*(*output object*)." When designers draw a finite state diagram on STG, every states and transitions' location can be changed by mouse dragging them to anywhere in the diagram. Each state or transition's name can be keyed in from keyboard. The input and output objects of each transition can be selected from a GIO object file to link the graphic interface. Similarly, designer can select the reusable software ICs in software IC libraries for getting input and output functions. If the functions what they need do not exist in the libraries, they can add their own specific software ICs into the software IC libraries.

The STG creates a table to record all variables of transitional function δ, including current state q_{i-1}, input object x_i, input function f_i, output object y_i, output function g_i, and target state q_i. The table is called finite state table. It can be seen that the number of transitional functions in a finite state diagram is the number of transition rows in the generated finite state table. The relationship between a finite state diagram and its finite state table are shown in Fig. 5 and TABLE I respectively. A designer draws a finite state diagram with three states "S1", "S2" and "S3" and two transitions. The finite state diagram links the input and output objects of transitions from a GIO object file, and gets input and output functions of transitions from software IC libraries. The information of this finite state diagram will transform into a finite state table file including the translated states, linked objects and needed functions.

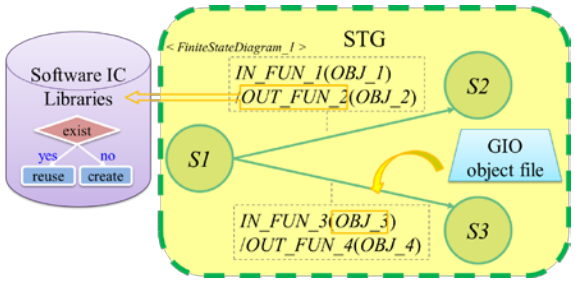


Fig. 5 A sample of finite state diagram.

TABLE I
THE FORMAT OF A SAMPLE OF FINITE STATE TABLE

Current State	*Input Object*	*Input Function*	*Output Object*	*Output Function*	*Target State*
S1	OBJ_1	IN_FUN_1	OBJ_2	OUT_FUN_2	S2
S1	OBJ_3	IN_FUN_3	OBJ_4	OUT_FUN_4	S3

C. *Software Processor*

The Software Processor is a universal kernel program which can execute any given state table. As shown in Fig. 6, the Software Processor is driven by a finite state table and links software ICs from software IC libraries. With the information of each GIO objects in a GIO object file, Software Processor builds graphic interface by creating a Windows Form and some Windows Form Control objects.

When the Software Processor is running, it records what the present state of the specific state table is. The present state at the beginning of running is the unique initial state of the finite state table. The Software Processor waits until any reasonable input is received. When receiving an possible input

from an input object via a specific input function, the Software Processor checks the present state and the reasonable input on every transition row, including its current state, input object, and input function. If all those above are matched, the Software Processor changes the present state into the target state of the matched transition row and activates the matched output function of the corresponding output object. Because the input and output function are all selected software ICs, it just likes that the Software Processor connects the software ICs together according to the information of the finite state table to realize designers' requests.

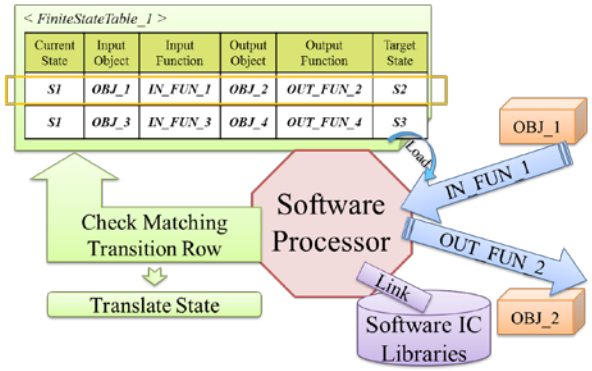
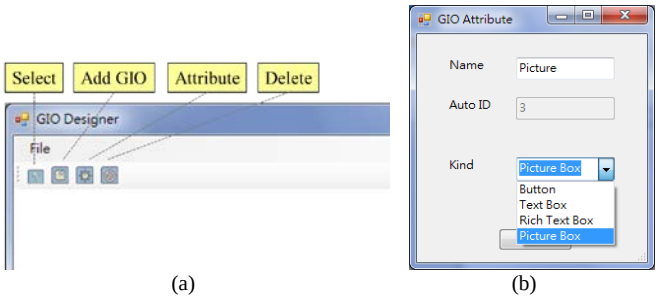


Fig. 6 The role of the Software Processor and its activity.

III. The SDP Toolset

The development environment of SDP is Microsoft visual studio 2008 with Microsoft.NET framework. The developing language is C-Sharp (or C#).

The user interface of the GIOD is shown in Fig. 7(a). GIOD is a simple tool to produce input and output objects for Windows GUI programming. Designers can design what their Windows Form program looks like and "select," "add " and "delete" each GIO object on the GIOD by the mouse. If clicking the button "Attribute," designers can select a Windows Form Control type for each GIO object as shown in Fig. 7(b). The pull down menu button "File" is used to save all the GIO objects into a GIO object file.

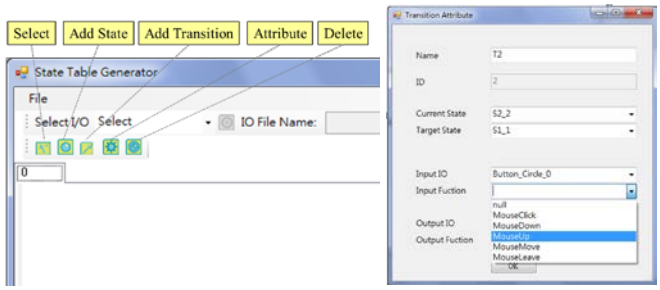


(a) (b)

Fig. 7 The user interface of GIO Designer. (a) The user interface. (b) The GIO object attribute form.

Fig. 8(a) is the snapshot of the proposed STG tool. STG lets designers draw finite state diagrams on the underside of the tool to explain their requests and transforms each diagram into a finite state table file. The pull down menu button "File" is used to save the finite state diagrams which designer design and the transformed finite state tables.

Every state diagram which is designed by designers has to have one unique initial state which has a small green rhombus at the upper left corner and one or more final states which has additional concentric circle. When the "Attribute" button being clicked and selecting any state or transition on the diagram, STG shows the Windows Form "State Attribute" or "Transition Attribute" as depicted in Fig. 8(b). Each state or transition has its own unique identification, named "ID," as one of its attributes for making off other states and transitions. The current state and target state can be changed by selecting another state on the finite state diagram, and the selected transition arrow will change and redraw in the same time. Designer can also select the GIO objects they need to be the input objects or output objects with the GIO object file generated from the GIOD. The selectable input and output functions are the exiting executable functions in software IC libraries which builds by designers.



(a) (b)

Fig. 8 The user interface of State Table Generator. (a) The user interface. (b) The transition attribute form.

To translate a state to another state needs at least one transition between states. Because one transition can only have one output function, the input function and input object of transition can be "*NULL*" at the same time for absolutely translating state. At the beginning, the absolutely translations usually set for the initial setting of a program. Likewise, one transition can only have one input function, and the output function and output object of a transition also can be "*NULL*" at the same time for another chance to receive input functions.

For solving the problem of reusing existing executable input functions and output functions of transitions which can be selected, we build software IC libraries outside of STG. The software IC libraries use the concept of dynamic-link library (DLL) [6, 10], which is an executable file that acts as a shared library of functions. Each function in the software IC libraries is written in a specific format designed by this work for using by the Software Processor. The STG gets the input and output functions' name by linking the software IC libraries. Designers can select the input and output functions they need or create the new useful functions here.

The last step is letting the finite state table generated by the STG drives the Software Processor. The Software Processor includes a history recorder for checking the correctness of

each finite state table. A sample record of the history recorder is shown in Fig. 9. The button "Load" on the toolbar is used to select a specific finite state table. After selecting a finite state table and starting the program, the text box of the recorder with a title string "--- History ---" records each step of state translation of the selected finite state table, including "table name," "present state," "input object," "input function," "matched transition name," "matched output object," "matched output function" and "matched target state". Designer can trace the operations of the specific program by the recorded history. The Software Processor builds the graphic interface according to the information of each GIO object. Because the existing executable software ICs in the software IC libraries is written by a specific format, the Software Processor can concatenate the needed functions based on the selected finite state table.

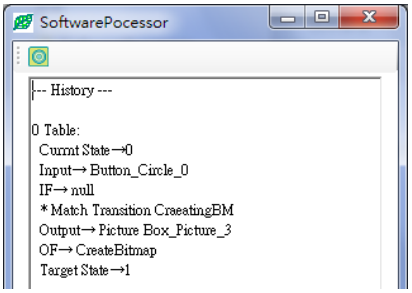


Fig. 9 A sample output of the history recorder.

IV. Applying the Platform

To demonstrate SDP, a Windows Form program "Drawer" is employed as an example. It looks like a simplified Microsoft Paint, and it can draw circles, rectangles and filled circles. The interface has three Windows buttons which can select what shapes users want to draw, and a Windows picture box which can display what users drew.

Designers use GIOD to design the graphic interface of the program. Because of the requests of the Drawer, as shown in Fig. 10, there are four GIO objects including three buttons and a picture box on GIOD. The size and location of each selected GIO object can be created, deleted and moved with the mouse. The type can be set with "GIO Attribute Form", and every type of each GIO object has a unique color on it.

The finite state diagram on STG for the Drawer is shown in Fig. 11. Fig. 11 shows that the initial state "*Initial*" absolutely translates to the second state "*Canvas Created*" and creates a new canvas on picture box "Picture" of the GIO objects at the beginning. If one of the three buttons is clicked, the state will be translated to one of the three drawing-states "*Button Circle Clicked*," "*Button Rectangle Clicked*" and "*Button Fill Circle Clicked*" depending on which button is pressed. For example, after clicking the button "Rectangle," the state will be changed to the state "*Button Rectangle Clicked*."

When the present state is one of the three drawing-states, two cases are necessary to be consider for the state transition. In the first case, if any of these three buttons is clicked, the state is changed to one of the three drawing-states depending on which button is clicked. In the second case, if the left button of the mouse is pressed on the canvas, the present state is changed to one of the three getting-mouse-location-states, "*Get Circle First Point*", "*Get Rectangle First Point*" and "*Get Fill Circle First Point*" depending on which the present state is. For example, when the present state is "*Button Rectangle Clicked*" and the button "Circle" is clicked, the state is changed to "*Button Circle Clicked*." When the present state is "*Button Circle Clicked*" and the user presses the left button of the mouse on the canvas, the present state is changed to "*Get Circle First Point*." Out of the above two cases, the present state will not be changed.

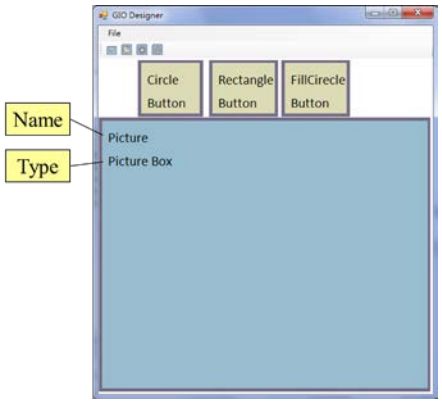


Fig. 10 An example "Drawer" on the GIOD.

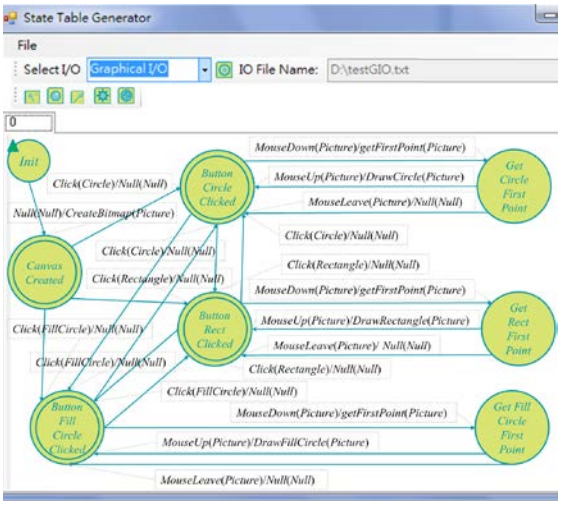


Fig. 11 The example "Drawer" on the STG.

When the current state is one of the three getting-mouse-location-states, two cases should be discussed about changing the state. In the first case, if the left button of the mouse is released, a simple graph, such as a rectangle or a circle, is drawn on the canvas, and the present state goes back to the previous drawing-state. In the second case, if the cursor leaves the canvas, the present state is changed back to the previous drawing-state and the kernel program activates a null function which means that nothing will be outputted.

TABLE II
THE FINITE STATE TABLE OF THE DRAWER

Current State	*Input Object*	*Input Function*	*Output Object*	*Output Function*	*Target State*
Init	Null	Null	Picture	CreateBit-map	Canvas Created
Canvas Created	Circle	Click	Null	Null	Button Circle Clicked
Canvas Created	Rec-tangle	Click	Null	Null	Button Rect Clicked
Canvas Created	Fill-Circle	Click	Null	Null	Button FCircle Clicked
Button Circle Clicked	Rec-tangle	Click	Null	Null	Button Rect Clicked
Button Circle Clicked	Fill-Circle	Click	Null	Null	Button FCircle Clicked
Button Circle Clicked	Picture	MouseDown	Picture	getFirstPoint	Get Circle First Point
Button Rect Clicked	Circle	Click	Null	Null	Button Circle Clicked
Button Rect Clicked	Fill-Circle	Click	Null	Null	Button FCircle Clicked
Button Rect Clicked	Picture	MouseDown	Picture	getFirstPoint	Get Rect First Point
Button FCircle Clicked	Circle	Click	Null	Null	Button Circle Clicked
Button FCircle Clicked	Rec-tangle	Click	Null	Null	Button Rect Clicked
Button FCircle Clicked	Picture	MouseDown	Picture	getFirstPoint	Get Fill Circle First Point
Get Circle First Point	Picture	MousebtUp	Picture	DrawCircle	Button Circle Clicked
Get Rect First Point	Picture	MouseUp	Picture	DrawRectan gle	Button Rect Clicked
Get Fill Circle First Point	Picture	MouseUp	Picture	DrawFill-Circle	Button FCircle Clicked
Get Circle First Point	Picture	MouseLeave	Null	Null	Button Circle Clicked
Get Rect First Point	Picture	MouseLeave	Null	Null	Button Rect Clicked
Get Fill Circle First Point	Picture	MouseLeave	Null	Null	Button FCircle Clicked

With the STG, designers can select one of GIO object files for every finite state diagram for linking the names of input and output objects of each transition. For example, at the top of Fig. 11, the designer selects the GIO object file "*testGIO.txt*." The information of those GIO objects in this GIO object file will be added into the database of the present finite state diagram. The names of those GIO objects are selected for communicating with appropriate graphic interface by designers. After finishing the finite state diagram, STG transforms the finite state diagram to a finite state table. TABLE II shows the transformed finite state table of "Drawer." Each transition row in the finite state table are transformed by each transition in the finite state diagram. Software Processor is driven by TABLE II. Fig. 12 demonstrates the result of the foregoing example. Users can use the produced program to draw some rectangles and circles.

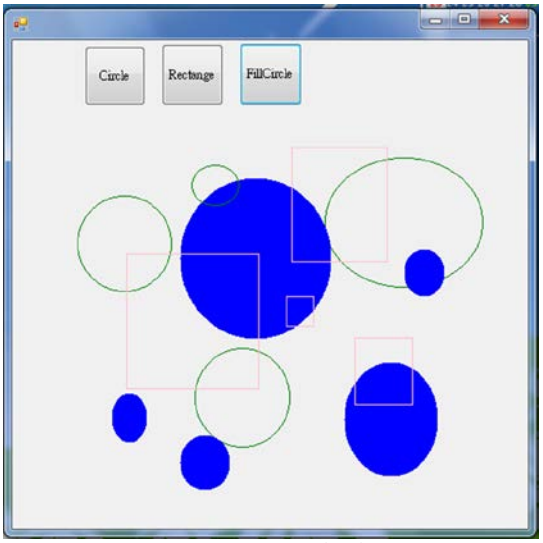


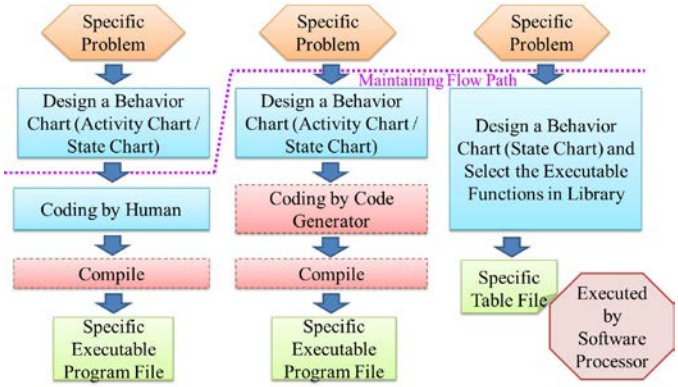
Fig. 12 The Drawer outcome of Software Processor.

V. COMPARISON OF DESIGN APPROACHES

In order to show the productivity improvement of this work, the universality of users, the time taken on development and the ease of maintenance of previous traditional design method, code generator design method and SDP are compared in this section. The designing flow path of the traditional design method is shown in Fig. 13(a). A software engineer usually draws a behavior chart to help him to describe the target program. After that, the engineer has to code the program by himself to get source codes. The source codes will be compiled to an executable program to solve the problem with a operating system or reasonable resources. The code generator design method, for example STATEMATE or Rhapsody, as shown in Fig. 13(b). Software engineers do not need to code any program by themselves anymore and the source codes are generated by a code generator automatically. The same thing with traditional design method is that the products of code generator are the source codes, too. Furthermore, the code

generator design method needs a very large library for generating each kind of programming languages.

The Fig. 13(c) is shown about the flow path proposed by this work. The strong point of this method is that the using functions in this method are all existing executable functions. With these functions (Software ICs) and the Software Processor, engineers can skip the coding step. Therefore, if an engineer wants to design a new program for a specific problem, he even has the possibility of finishing the program without coding any function. The products of this work are finite state tables rather than source codes. The emphasis of maintaining flow paths is that these flow paths are running very often. That is to say, the recompile time and the recoding time of traditional design method and code generator design method are magnified so much so that can't be ignored.



(a) (b) (c)

Fig. 13 The designing flow path of (a) the traditional design method, (b) code generator design method and (c) SDP.

TABLE III
COMPARISON OF DESIGN APPROACHES

	Traditional Design Method	***Code Generator Design Method***	***SDP***
Designers Understanding Programming Languages	Yes	Probability Not	Probability Not
Specially Extra Libraries	No	For Transforming Behavior Diagrams to Specific Codes	For Storing Software ICs
Maintaining Pattern	By Programing Codes	By Behavior Diagrams	By Finite State Diagram
Recoding and Recompiling	Yes	Yes	No
Efficiency of Executing Programs	High	High	Low

TABLE III shows the difference between traditional, code generator and SDP methods' properties more clearly. From the first row of this table, one of the advantages of SDP is the function reuse with the reusable software ICs. Therefore, if the necessary functions of target program have already been created in software IC libraries, designers can design a program without learning any programming language. Similarly, SDP skip the coding time, so they also are not constrained by general languages, too. They can just select the appropriate functions with the names or footnotes in the finite state diagrams which they designed on STG. Furthermore, the reusable software ICs can reduces the developing time obviously. Another good quality of SDP is visually design and maintenance which can reduce design and maintenance time. The products of SDP are finite state tables, so the designers do not need to compile.

The weakness of the approach is the Software Processor checks the present finite state table when receive any input. Because of that, the efficiency of executing program has a few decrease.

VI. CONCLUSIONS

Improving software productivity is the most important issue in software engineering. Software product maintenance always costs too much time and manpower. This work designs and implements a software development platform for generating programs efficiently and making maintenance easily. With the provided GIOD, STG, Software Processor and some libraries, software designers can visually produce and maintain any program by designing and refining a finite state diagram.

REFERENCES

[1] K. Buchenrieder and C. Veith, "A prototyping environment for control-oriented HW/SW systems using state-charts, activity-charts and FPGA's", In: *Proceedings of the 1994 conference on European design automation*, Sep. 1994, pp. 60–65.

[2] G. Booch, J. Rumbaugh and I. Jacobson, *The Unified Modeling Language User Guide 2e*, Addison-Wesley, CA, 1999.

[3] E. Gery, D. Harel, and E. Palatshy, "Rhapsody: A complete life-cycle model-based development system", In: *Proceedings of the Third International Conference on Integrated Formal Methods*, 2002, pp. 1–10.

[4] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, no. 3, pp. 231–274, Jun. 1987.

[5] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-trauring, and M. Trakhtenbrot, "STATEMATE: A working environment for the development of complex reactive systems", *IEEE Transactions on Software Engineering*, vol. 16, no. 4, pp. 403–414, Apr. 1990.

[6] J.M. Hart, *Windows System Programming 3/e*, Addison-Wesley, CA, 2005.

[7] J.E. Hopcroft, R. Motwani and J.D. Ullman, *Introduction to Automata Theory, Languages and Computations 2/e*, Addison-Wesley, NJ, 2001.

[8] J. Koskinen, *Software Maintenance Costs*, http://users.jyu.fi/~koskinen/smcosts.htm, Sep. 10, 2010.

[9] P. Linz, *An Introduction to Formal Languages and Automata 4/e*, Jones & Bartlett, Mississauga, Canada, 2006.

[10] Microsoft, *Microsoft Developer Network (MSDN)*, http://msdn.microsoft.com/en-us/library/1ez7dh12(v=VS.90).aspx.

[11] Object Management Group, *Unified Modeling Language Version 2.3*, http://www.omg.org/spec/UML/2.3, May 2010.

[12] M.O. Rabin and D. Scott, "Finite automata and their decision problems," *IBM Journal of Research and Development*, vol. 3, no. 4, pp. 114–125, Apr. 1959.

[13] R.W. Selby, *Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research*, Wiley-IEEE Computer Society Press, CA, Jun. 2007

Security Specifications for a Multi-disciplinary Research Project

Syed (Shawon) M. Rahman, Ph.D.
Assistant Professor, Department of Computer Science and Engineering, University of Hawaii-Hilo
200 W. Kawili St, Hilo, HI 96720 USA
SRahman@Hawaii.edu

Michael Peterson, Ph.D.
Assistant Professor, Department of Computer Science and Engineering, University of Hawaii-Hilo
200 W. Kawili St, Hilo, HI 96720 USA
mrp2@Hawaii.edu

***Abstract* - Many organizations are not able to protect their assets from hackers or attackers until their systems or businesses are affected. The security specification process provides an organization with a proactive approach to operate in the most cost efficient manner with a known and acceptable level of security risk. In this paper, we have addressed IT security specifications for a multi-disciplinary large scientific project and identified assets, common threats, and possible vulnerabilities. Security specifications also give organizations a consistent, clear path to categorize and prioritize limited resources in order to manage security risks.**

Keywords- Software Security, Security Specifications, Secure System, Risk Assessment, Threats, Vulnerabilities, and Risk Analysis.

I. INTRODUCTION

Most organizations understand the critical role that IT plays in underneath their business objectives; however, today's highly connected IT infrastructures exist in an environment that is increasingly hostile—attacks are being mounted with increasing incidence and are demanding ever shorter response times [1]. In practice, many organizations are not able to protect their assets from new security threats until their businesses are impacted. In this paper, we discuss security risk specifications for a multi-disciplinary scientific research project. We propose the real-time remote visualization architecture in Figure 1. Visualization, data modeling, high performance computing, scientific computation, etc. are major components of our project. We aim to provide a proactive approach to protect our assets from attackers. We have selected our current NSF funded project as a case study; however, this information is useful to most organizations' research projects. We aim to adopt GPU (Graphics Processing Unit) based Visualization's tools and technologies in our project. Our goal is to utilize GPU based technology in teaching and students learning. We believe visualization can reveal things otherwise hidden and very difficult to understand. Visualization can assist to summarize complicated information, can illustrate relationships that might otherwise be hidden, and can increase classroom engagement, retention, and overall students' learning performance [9].

II. BACKGROUND STUDY

A. Why Security is Crucial?

Information security is very crucial in this project and we need to take proper protection to prevent possible attacks (both inside and outside) to protect our resources. The United States Computer Emergency Readiness Team (US-CERT) [10] interacts with federal agencies, industry, the research community, state and local governments, and others to collect reasoned and actionable cyber security information and to identify emerging cyber security threats. Based on the information reported, US-CERT identifies the following cyber security trends (figure 2 and 3) for fiscal year 2009 first quarter [10].

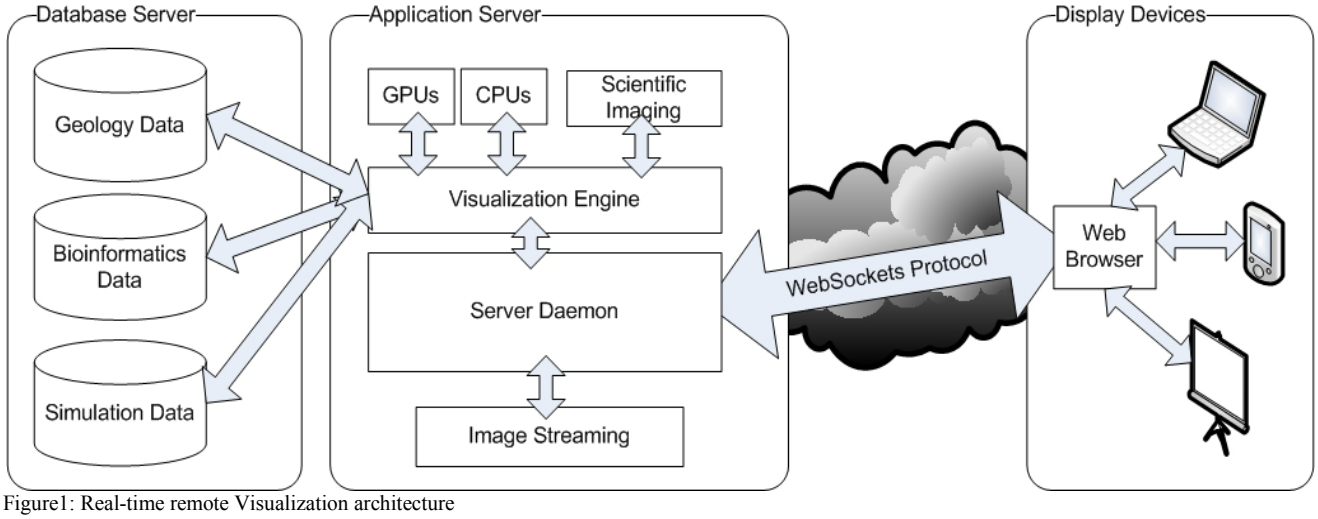


Figure1: Real-time remote Visualization architecture

Figure 2 displays the overall distribution of cyber security incidents and events across six major categories. At the time of their study, the percentage of Category 5 (Scans, Probes, or Attempted Access) reports decreased for the second consecutive quarter. This was a 2.9% decrease in CAT 5 incidents compared to the previous quarter. The percentage of Malicious Code incidents increased by 3.3%.

Figure 3 is a breakdown of the top five incidents and events versus all others. Phishing remained the most prevalent incident type, accounting for 70% of all incidents reported. This was a slight percentage decrease of 1.8% from the previous quarter. On the other hand, The sophistication of attack tools has gone up, while the level of skill required to use those tools has gone down (see figure 4). At this stage, the attacker takes advantage of his or her ability to steal confidential and proprietary data and sells it for profit or uses it for military intelligence [12].

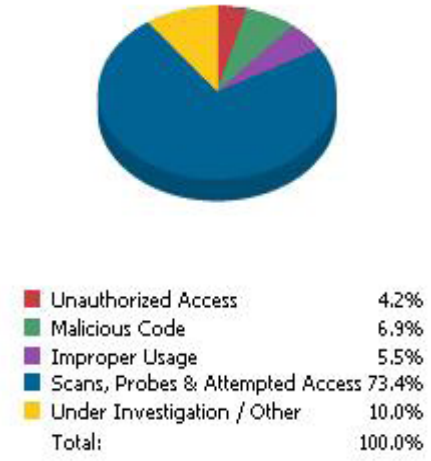


Figure 2: Incidents and Events by Category [11].

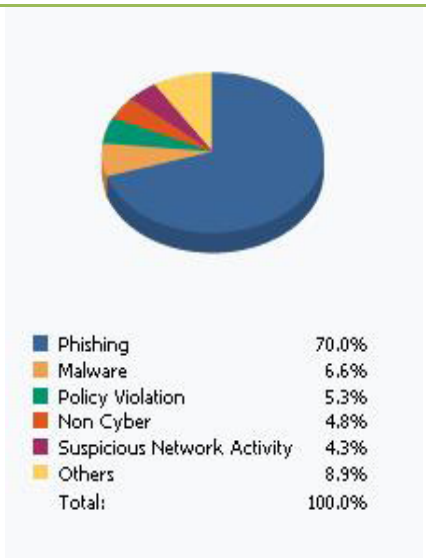


Figure 3: Top Five Incidents vs. All Others [11].

B. Security Vulnerability Trends for 2011 and beyond

In this section, we discuss the security vulnerability trends in 2011 and beyond. We anticipate that our research project will encounter similar threats. Hackers and malware spreaders demonstrate increasing sophistication, meaning computer users need to become wiser and more proactive, too. Whether an individual uses a mobile phone to check e-mail and surf the Web or an enterprise IT administrator works to safeguard our data, Symantec identifies the following 13 security issues most relevant for 2010 [2][11] (the following points are paraphrased from source [11]):

- **Antivirus is not enough**: We have reached an inflection point, where new malicious programs are actually being created at a higher rate than good programs. Approaches to security that looks for ways to include all software files, such as reputation-based security, are key in 2011 [**11**].
- **Social engineering as the primary attack vector**: More and more, attackers are going directly after the end user and attempting to hoax them into downloading malware or divulging sensitive information under the auspice that they are doing something perfectly innocent [11].
- **Rogue security software vendors escalate their efforts**: In 2011, Symantec expect to see the propagators of rogue security software scams take their efforts to the next level, even by hijacking users' computers, rendering them useless and holding them for their financial gain.
- **Social networking third-party applications will fraud targets**: With the popularity of social networking sites poised for another year of unprecedented expansion, expect to see fraud being targeted toward social site users to raise [11].
- **Windows 7 has come in the crosshairs of attackers**: Microsoft's new operating system is no exception, and as Windows 7 gains traction in 2011, it is almost certain that attackers will find more ways to exploit its users [11].
- **Fast Flux botnets will increase**: As industry countermeasures continue to reduce the effectiveness of traditional botnets, Symantec expects to see more using this method to carry out attacks.
- **URL-shortening services become the phisher's best friend**: In an attempt to evade anti-spam filters through obfuscation, expect spammers to use shortened URLs to carry out their evil deeds[11].
- **Mac and Mobile Malware Will Increase**: because Mac and Smartphone's continue to increase in popularity in 2011, more attackers will devote time to creating malware to exploit these devices[11].
- **Spammers breaking more rules**: since the economy continues to suffer and more people seek to take advantage of the loose restrictions of the Federal Trade Commission's Can-Spam Act, there will be more organizations selling unauthorized e-mail address lists and more less-than-legitimate marketers spamming those lists[11].
- **As spammers adapt, volume will continue to**: Since 2007, spam has increased on average by 15 percent a year. Spam volumes will continue to fluctuate in 2011 as spammers continue to adapt to the sophistication of security software and the intervention of responsible ISPs and government agencies across the globe [11].
- **Specialized malware on the rise**: Highly specialized malware was uncovered in 2009 that was aimed at exploiting certain ATMs, indicating a degree of insider knowledge about their operation and how they could be exploited. Expect this trend to continue in 2011, including the possibility of malware targeting electronic voting systems, both those used in political elections and public telephone voting, such as that connected with reality television programs and competitions [11].
- **CAPTCHA [**13] **technology will improve**: This will prompt more businesses in emerging economies to offer real people employment to manually generate accounts on legitimate Web sites. Symantec estimates that the individuals will be paid less than 10 percent of the cost to the spammers, with the account farmers charging $30-$40 per 1,000 accounts [11].

- **Instant messaging spam will surge**: By the end of 2010, Symantec predicts that one in 300 IM messages will contain a URL. Also, in 2010, Symantec predicts that one in 12 hyperlinks overall will be linked to a domain known to be used for hosting malware. IM threats will largely be comprised of unsolicited spam messages containing malicious links, especially attacks aimed at compromising legitimate IM accounts [11].
- **Attack in Smartphones**: In 2011 there will be new attacks on mobile devices such as cell phones. Most of the existing threats target devices with Symbian, an operating system which is now on the wane. Of the emerging systems, PandaLabs predicts that the threats for Android will increase considerably throughout the year, becoming the number one mobile target for cyber-crooks [18].
- **Cyber war**: Stuxnet and the WikiLeaks cables suggesting the involvement of the Chinese government in the cyber-attacks on Google and other targets have marked a turning point in the history of these divergences. Stuxnet was an attempt to interfere with processes in nuclear plants, specifically, with uranium centrifuge. Attacks such as these, albeit more or less sophisticated, are still ongoing, and will undoubtedly increase in 2011, even though many of them will go unnoticed by the general public [18].
- **HTML5 Applications**: HTML5 is the perfect target for many types of criminals and could eventually replace Flash. It can be run by browsers without any plug-ins, making it even more attractive to find a security hole that can be exploited to attack users regardless of which browser they use [18].

III. CASE STUDY: EPSCOR-HAWAII FUNDED RESEARCH PROJECT

We highlight our current NSF EPSCoR-Hawaii research project as a case study. The primary focus of our current project is to enhance the sustainability of Hawaii's science and technology capabilities for understanding and predicting how invasive species, anthropogenic activities and climate change impact the biodiversity, ecosystem function and current or potential human use of Hawaiian focal species. A significant focus will build on the foundation created through prior EPSCoR awards, by expanding competitiveness in new areas of science and technology research and education, and improving relationships of the academic research community with the local community. Integral components of this Initiative include increasing capacity for CyberInfrastructure, data visualization and modeling, as well as broadening the diversity of the State's science, technology, engineering and math (STEM) workforce through enhancement of hands-on research experiences for undergraduate and graduate students in concert with broader community outreach. Researchers or stakeholders of this project are located in different states in the USA and their varied research backgrounds include Computer Science, Biology, Geography, Forestry, Marine Science, Astronomy, Physics, Mathematics, Geology, Geophysics, and so on. Currently, more than 30 active researchers are involved in this five-year long $20 Million project [2].

To enable visualization, scientific computation, modeling, etc. research in our project, we aim to establish a GPU Based Visualization Server as a three-tier web application (Figure 5). Our visualization server will meet the following desired features:

- Perform real-time scientific data visualization on large data sets.
- Support CAVE and 3D rendering though high definition video streaming.
- Perform scientific computing such as bioinformatics, simulation, modelling, and data mining.
- Host a web portal that will provide scientists and researchers from direct access and upload large data sets, perform scientific computing, and visualization.

As our visualization server is a 3-tier web application we predict it will encounter many similar threats or risks similar to other web applications. In section 4 we briefly discuss sample security specifications for our project.

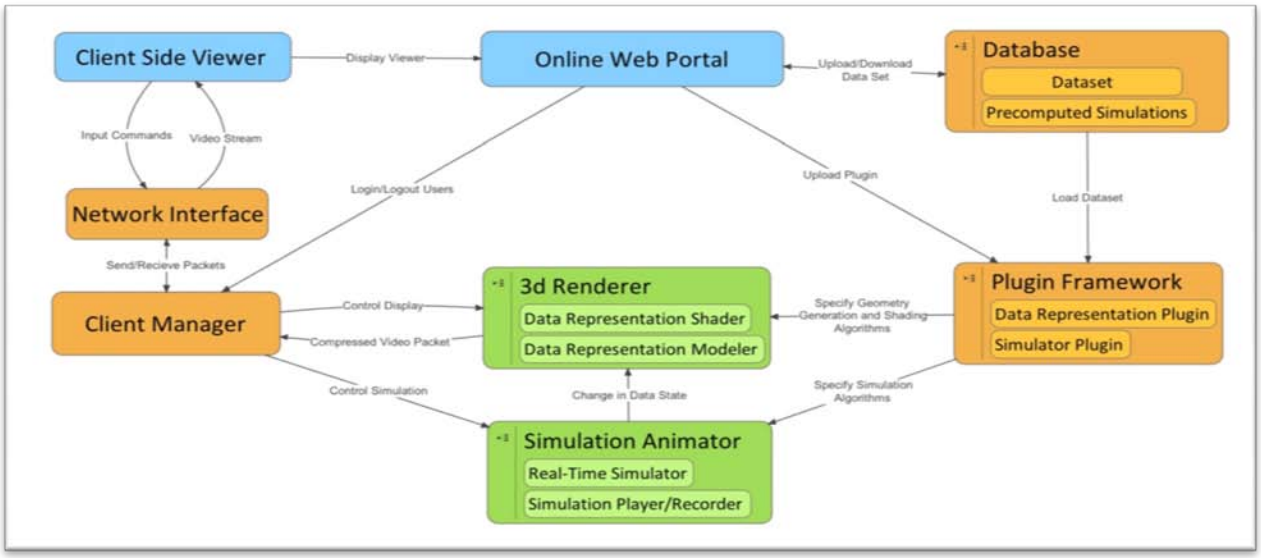


Figure 5. Visualization & scientific computation server architecture

IV. SECURITY SPECIFICATIONS

In this section, we specify security issues involved in our project. These specifications describe our project strategy for addressing security within our project scopes. They define a security model that supports, integrates, and unifies several popular security models, mechanisms, and technologies (including both symmetric and public key technologies) in a way that enables a variety of systems to securely interoperate in a platform- and language-neutral manner

We adapt the following security requirements in our project [5]:

- Ensure that access by users and client applications is controlled.
- Ensure that users and client applications are identified.
- Ensure that their identities are properly verified.
- Ensure that users and client applications can only access data and services for which they have been properly authorized.
- Detect attempted intrusions by unauthorized persons and client applications.
- Ensure that unauthorized malicious programs (e.g., viruses) do not infect the application or component.
- Ensure that communications and data are not intentionally corrupted.
- Ensure that parties to interactions with the application or component cannot later repudiate their interactions.
- Ensure that confidential communications and data are kept private.
- Enable security personnel to audit the status and usage of the security mechanisms.
- Ensure that applications and centers survive attack, possibly in degraded mode.
- Ensure that centers and their components and personnel are protected against destruction, damage, theft, or surreptitious replacement (e.g., due to vandalism, sabotage, or terrorism).
- Ensure that system maintenance does not unintentionally disrupt the security mechanisms of the application, component, or center.

To meet the above objectives, we will specify briefly each of the following equivalent kinds of security requirements [5]:

- Identification Requirements: It specifies the extent to which a business, application, component, or center shall identify its externals (e.g., human actors and external applications) before interacting with them.
- Authentication Requirements: It specifies the extent to verify the identity of its externals (e.g., human actors and external applications) before interacting with them.
- Authorization Requirements: It specifies the access and usage privileges of authenticated users and client applications.
- Immunity Requirements: It specifies the extent to which an application or component shall protect itself from infection by unauthorized undesirable programs (e.g., computer viruses, worms, and Trojan horses).
- Integrity Requirements: It specifies the extent to which an application or component shall detect and record attempted access or modification by unauthorized individuals.
- Intrusion Detection Requirements: It specifies the extent to which an application or component shall detect and record attempted access or modification by unauthorized individuals.
- Non-repudiation Requirements: It specifies the extent to prevent a party to one of its interactions (e.g., message, transaction) from denying having participated in all or part of the interaction
- Privacy Requirements: It specifies the extent to keep its sensitive data and communications private from unauthorized individuals and programs
- Security Auditing Requirements: It specifies the extent to which a business, application, component, or center shall enable security personnel to audit the status and use of its security mechanisms
- Survivability Requirements: It specifies the extent to which an application or center shall survive the intentional loss or destruction of a component.
- Physical Protection Requirements: It specifies the extent to which an application or center shall protect itself from physical assault.
- System Maintenance Security Requirements: It the extent to which an application, component, or center shall prevent authorized modifications (e.g., defect fixes, enhancements, updates) from accidentally defeating its security mechanisms [5].

Firesmith [5] has addressed the need to systematically analyze and specify real security requirements as part of the quality requirements for a business, application, component, or center.

V. RISK ASSESSMENT AND RISK ANALYSIS

The risk assessment and risk analysis processes used in security requirements specifications [2] [4]. We have adopted the following process in our multi-disciplinary research project by customizing the list provided by Stallings [3].

- Asset identification: We have identified the key system assets (or services) that have to be protected. An asset is "anything which needs to be protected" because it has value to our project and organization. Asset contributes to the successful attainment of the organization's objectives, and may be either tangible or intangible
- Asset value assessment: We have estimated the value of the identified assets.
- Exposure assessment: We have assessed the potential losses associated with each asset; however, it is not included in this paper.
- Threat identification: We have identified the most probable threats to our project assets.
- Attack assessment: We have decomposed threats into possible attacks on the system and the ways that these may occur so that we can learn more about the attacks and can prevent from attacks in future.
- Control identification: We have proposed a few controls that may be put in place to protect our assets; however, it would be an on-going process.
- Feasibility assessment: We have also performed some feasibility studies to assess the technical feasibility and cost of the controls.

To perform the above risk assessment process, we must identify our project assets, common threats, vulnerability, etc. carefully. We have identified our major / significant assets, possible major threat, and potential vulnerabilities.

A. Identify Assets

In following table (Table 1), we have listed high level of description of our project asset and their definition [1][2][3].

TABLE 1: MOST VALUABLE ASSETS [1], [2]

Overall IT Environment	Asset Name
Physical infrastructure	Data centers, Servers, Desktops, mobile devices, supercomputer clusters
Physical infrastructure	Server/End-user application software, Development tools, Routers, switches
Intranet data	Source code, Human resources data, Financial data, Employee passwords, Intellectual property, Employee profile information
Core infrastructure	Active Directory service, subversion
Core infrastructure	DNS, DHCP, Active directory, storage, file sharing, management tools, VPN,

TABLE 2: COMMON THREATS IN OUR PROJECT[1],[2]

Threat	Example
Catastrophic incident	Fire, flood, earthquake, storm, terrorist attack, landslide, avalanche, volcanic eruption
Mechanical failure	Power outage, Hardware failure, Hardware failure, Network outage
Malicious person	Hacker, cracker, Computer criminal, Social engineering, Dishonest employee, Malicious mobile code, Disgruntled former/current employee

B. Common Threats

In following table (Table 2), we have listed high level description of some threat and provided some examples of your project [1][2][3].

C. Vulnerabilities

In following table (Table 3), we have identified high level vulnerability class and provided brief description of each vulnerability [1][2][3].

VI. CONCLUSION

There are thousands of security risks, threats, and vulnerabilities already out there and every moment more and more sophisticated attacks are taking place around the world. Unless we take a proactive approach by specifying security risks for any organization, it is very difficult to protect our assets from hackers or attackers. Still many organizations are not able to guard their valuable resources until their systems or services affected by external or internal attackers. In this paper, we have specified security risks in our multi-disciplinary scientific project; however, this information should be generic enough to apply to most of the IT infrastructure.

TABLE 3: POSSIBLE VULNERABILITIES IN OUR PROJECT [1], [2]

Vulnerability Class	Vulnerability
Physical	Unlocked doors, Unguarded access to computing facilities,
Natural	Facility located in a volcanic eruption area
Hardware	Missing patches, Outdated firmware, Systems not physically secured,
Software	Out of date antivirus software, Missing patches, Poorly written applications (Cross site scripting, SQL injection, etc.)
Software	Deliberately placed weaknesses (Spyware such as keyloggers, Trojan horses, vendor backdoors, etc)
Communications	Unencrypted network protocols, Connections to multiple networks, Unnecessary protocols allowed
Human	Poorly defined procedures(Insufficient incident response preparedness, Violations not reported)

We have identified our project's most important assets, common threats, and possible vulnerabilities to utilize and prioritize our limited resources. We believe a formal security risk assessment and risk analysis process enables any organization to operate in the most cost efficient manner with a known and acceptable level of business risks and gives organizations a consistent and clear path to organize and prioritize limited resources in order to protect their valuable resources. We have concluded that web- based software, servers, network devices, cyber portals, user data, and mobile devices are the most vulnerable entities. We recommend that our research team should be proactive protecting our assets from malware and malicious users or systems. To do this we must have written IT security policies/guidelines and IT security specification in place.

REFERENCES

[1] Microsoft Corporation, "The Security Risk Management Guide", 2006 Microsoft Corporation. Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA. URL: http://www.microsoft.com/downloads/; Web Accessed on October 14, 2010.

[2] Rahman, Syed and et al; "IT Security Assessment for Interdisciplinary Research"; The First International workshop on Communications Security & Information Assurance (CSIA), In conjunction with the Second International

Conference on Wireless & Mobile Networks (WiMo-2010), June 26 -28, 2010, Ankara, Turkey.

[3] Stallings, William and Brown, Laurie; Computer Security: Principles and Practice, First Ed, Pearson Education, Inc, Upper Saddle River, NJ, 2008.

[4] Sommerville, Ian; Software Engineering, 9th edition, Addison Wesley, Boston, USA, 2011, Page(s): 329-340

[5] Liu, Simon and Cheng, Bruce; "Cyberattacks: Why, What, Who, and How", IT Pro, IEEE Computer Society, May/June 2009.

[6] Firesmith, Donald: "Engineering Security Requirements", in Journal of Object Technology, vol. 2, no. 1, January-February 2003, pp. 53-68.http://www.jot.fm/issues/issue_2003_01/column6

[7] Menn, Joseph; "Cyber-criminals breaching trusted websites", Financial Times Ltd, FT.com. London, Sep 15, 2009.

[8] Dong-Her Shih, Binshan Lin, Hsiu-Sen Chiang, & Ming-Hung Shih; "Security aspects of mobile phone virus: a critical survey"; Industrial Management + Data Systems, 108(4), 478-494. 2008

[9] Tom Crawford and Brandon Hall; "Visualization in Learning: 14 Case Studies That Emphasize Visual Thinking", [online] http://www.brandon-hall.com/publications/visualization/visualization.shtml, Web retrieve on October 12, 2010.

[10] US-CERT, "Cyber Security Trends, Metrics, and Security Indicators", June 16, 2009. Volume 4, Issue 1.http://www.us-cert.gov/press_room/trendsanalysisQ109.pdf

[11] Barrett, Larry; "Symantec's 'Unlucky 13' Security Trends for 2010", http://www.internetnews.com/security/article.php/3849371/Symantecs+Unlucky+13+Security+Trends+for+2010.htm, web retrieve on October 14, 2010

[12] Ahamed, Syed; Intelligent Internet Knowledge Networks, John Wiley and Sons, Hoboken, New Jersey 2006.

[13] Captcha, http://www.captcha.net/, web retrieve on October 14, 2010

[14] Pak, Charles and Cannady, James; " Asset priority risk assessment using hidden markov models", Proceedings of the 10th ACM conference on SIG-information technology education, Fairfax, Virginia, USA , Pges 65-73, 2009.

[15] Mullikin, Arwen and Rahman, Syed (Shawon); "The Ethical Dilemma of the USA Government Wiretapping"; International Journal of Managing Information Technology (IJMIT); ISSN : 0975-5586

[16] Rahman, Syed (Shawon) and Donahue, Shannon; "Convergence of Corporate and Information Security"; International Journal of Computer Science and Information Security, Vol. 7, No. 1, 2010; ISSN 1947-5500

[17] Bisong, Anthony and Rahman, Syed (Shawon); " An Overview of the Security Concerns in Enterprise Cloud Computing "; International journal of Network Security & Its Applications (IJNSA)ISSN: 0975 - 2307

[18] CIO Update, "Top 10 Technology Security Trends for 2011", December 14, 2010, web retreive on February 7, 2011 from http://www.cioupdate.com/research/article.php/3917131/Top-10-Technology-Security-Trends-for-2011.htm

Monitoring Errors in Integration Workflows

Rafael Z. Frantz
UNIJUÍ University
Department of Technology
Rua do Comércio, 3000, Ijuí, 98700-000, RS, Brazil
Email: rzfrantz@unijui.edu.br

Rafael Corchuelo
University of Seville
ETSI Informática
Avda. Reina Mercedes, s/n. Seville 41012. Spain
Email: corchu@us.es

Carlos R. Rivero
University of Seville
ETSI Informática
Avda. Reina Mercedes, s/n. Sevilla 41012. Spain
Email: carlosrivero@us.es

Carlos Molina-Jiménez
Newcastle University
School of Computing Science
Newcastle upon Tyne, NE1 7RU, United Kingdom
Email: carlos.molina@ncl.ac.uk

***Abstract*—Enterprise Application Integration (EAI) is a field of Software Engineering. Its focus is on helping software engineers integrate existing applications at a sensible costs, so that they can support new business processes or optimise existing ones. EAI solutions are distributed in nature, which makes them inherently prone to failures. In this paper, we report on a proposal to address error detection in EAI solutions. The main contribution is that it runs in linear time, it deals with both choreographies and orchestrations, and that it is independent from the execution model used.**

Keywords: Distributed systems; Enterprise Application Integration; Fault-tolerance; Error detection algorithms.

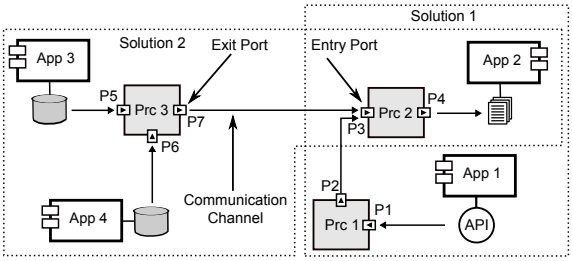


Figure 1. Sample EAI solutions.

I. Introduction

Companies are relying heavily on computer-based applications to run their businesses processes. Such processes must evolve and adapt as companies evolve and adapt to varying contextual conditions. Common problems include that the applications were not designed to facilitate integrating them with others, i.e., they do not provide a business level API, and that they were implemented using a variety of technologies that do not inter-operate easily. The goal of Enterprise Application Integration (EAI) is to help reduce the costs of EAI solutions to facilitate the implementation and evolution of business processes.

Figure §1 sketches two sample EAI solutions that involve four applications and three integration processes. Note that a solution is only a logical means to organise a set of processes: different solutions can share the same processes, and a solution can contain another solution. The processes interact with the applications using the facilities they provide, e.g., an API in the best case, a user interface, a file, a database or other kinds of resources. They help implement message-based workflows to keep a number of applications' data in synchrony or to build new functionality on top of them. Processes use ports to communicate with each other or with applications over communication channels. Ports encapsulate reading from or writing to a resource, which helps abstract away from the details of the communication mechanism, which may range from an RPC-based protocol over HTTP to a document-based protocol implemented on a database.

The Service Oriented Architecture initiative has gained importance within the field of EAI, since it provides appropriate technologies to wrap applications (so that they provide business APIs) and to implement message workflows. Centralised workflows, aka orchestrations, rely on a single process that helps co-ordinate a workflow of messages amongst a number of other processes and applications; contrarily, decentralised workflows, aka choreographies, do not rely on such a central co-ordinator. The tools used to implement workflows include conventional systems [1, 8], others based on BPEL, and others like BizTalk [5] or Camel [10].

EAI solutions are distributed in nature, since they involve several applications and processes that may easily fail to communicate with each other [8], which argues for real-world EAI solutions to be fault-tolerant. There seems to be a general consensus that the provisioning fault-tolerance includes the following stages: event reporting, error monitoring, error diagnosing, and error recovery. Event reporting happens when processes report that they have read or written a message by means of a port; the goal of error monitoring is to analyse traces of events to find invalid correlations, i.e., anomalous sets of messages that have been processed together; such correlations must later be diagnosed to find the cause of the anomalies, and appropriate actions to recover from the error

must be taken in the error recovery stage.

Orchestration workflows rely on an external mechanism that analyses inbound messages, correlates them, and starts a new instance of the orchestration whenever a correlation is found. The typical execution model is referred to as process-based since a thread must be allocated to run a process on a given correlation; contrarily, the task-based execution model relies on a pool of threads that are allocated to the tasks. Simply put, in the process-based model threads remain allocated to a process even if that process is waiting for the answer to a request to another process; contrarily, in the task-based model, no thread shall be idle as long as a task in a process is ready for execution.

In this paper, we report on a proposal to build an error monitor for EAI solutions. The key contribution is that it works with both orchestrations and choreographies, and that it is independent from the execution model used. In Section §II, we report on other proposals in the literature; in Section §III, we present an overview of our proposal; in Section §IV, we delve into our proposal to detect errors; in Section §V, we analyse our proposal both from a theoretical and a practical point of view; finally, we present our conclusions and future work in Section §VI; appendix §A provides a few ancillary proofs that support our theoretical analysis.

II. Related Work

Error detection is relatively easy in orchestration systems because either correlations are found prior to starting an orchestration process and everything happens within the boundaries of this process. Contrarily, in choreographies, a correlation may typically involve several processes that run in total asynchrony, and there is not a single point of control; furthermore, EAI solutions may overlap since it is common that processes are reused across several business processes. This makes it more difficult to endow choreographies with fault-tolerance capabilities.

The research on fault tolerance that has been conducted by the workflow community is closely related to our work. Chiu and others [4] presented an abstract model for workflows with embedded fault-tolerance capabilities; it set the foundations for other proposals in this field. Hagen and Alonso [8] presented a proposal that builds on the two-phase commit protocol, and it is suitable for orchestrations in which the execution model is process-based. Alonso and others [1] provided additional details on the minimum requirements to deal with fault tolerance in orchestrated systems. Liu and others [12] discussed how to deal with fault tolerance in settings in which recovery actions are difficult or infeasible to implement; the authors also assume the existence of a centralised workflow engine, i.e., they also focus on orchestrations. Li and others [11] reported on a theoretical solution that is based on using Petri nets; they see processes as if they were controllers, and report on detecting some classes of errors by means of linear parity checks; the key is that they focus on systems in which a fault can involve an arbitrarily large number of correlated messages, which are consumed and produced by distributed processes, but are assume that they are choreographed by a central processor. An architecture for fault-tolerant workflows, based on finite state machines that recognise valid sequences of messages was discussed in [6]; this proposal is suitable for both orchestrated and choreographed processes; however it is aimed at process-based executions.

The study of fault tolerance in the context of choreographies has been paid less attention in the literature. Chen and others [3] presented a proposal that deviates from the previous ones in that their results can be applied to both orchestrations and choreographies. They assume that the system under consideration is organised into three logical layers (front-end, application server, and database server), plus an orthogonal layer (the logging system). Since they can deal with choreographies, they need to analyse message traces to detect errors. They assume that each message has a unique identifier that allows to trace it throughout the execution flow; unfortunately, they cannot deal with EAI solutions in which messages are split or aggregated, since this would require to find correlations amongst messages, which is not supported at all. Due to this limitation, it can easily deal with both process- and task-based execution models. Yan and Dague [15], Yan and others [14] suggested to re-use the body of knowledge about error detection in industrial discrete event systems, in error detection in web services applications; they discussed runtime error detection of orchestrated web services; a salient feature of this proposal is that, similarly to [13], the authors assume that failure events are not observable; the granularity of execution in this approach is at process level. Baresi and others [2] discussed some preliminary ideas for building an error monitor that can be used for both orchestrated and choreographed processes. No implementation or evaluation was provided.

Our analysis of the literature reveals most authors in the EAI field focus on orchestrations and the process-based execution model; choreographies and the task-based execution model have been paid little attention so far. Another conclusion is that the distinction amongst the stages required to provision fault tolerance is often blurred. The reason is that many proposals focus on error recovery since error detection or error diagnosing is quite a trivial task. In many proposals, the presence of an error can be derived from a single event. For instance, the conventional try-catch mechanism involves the notification of a single event to be caught by the exception mechanisms [7]. However, there is a large class of applications in which the presence of an error can only be deduced from the analysis of traces of events that are related to each other, e.g., by order, parent-child relationships, propagation, or causations. Error detection in these cases is a challenging problem, in particular, when the number of events is large.

III. Overview of our Proposal

Our proposal builds on a monitor to which each port must report events, and a set of rules that help determine if correlations are valid or not, cf. Figure §2. A monitor is composed of three modules called Registrar, Event Handler, and

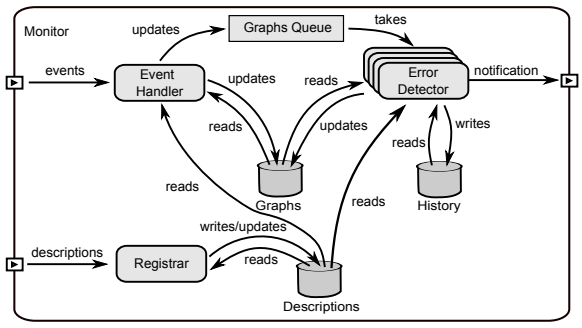


Figure 2. Structure of the monitor.

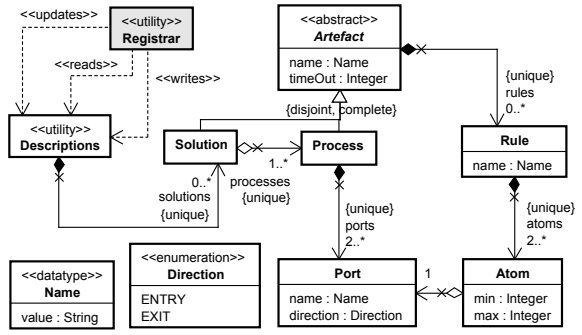


Figure 3. Model of the Registrar module.

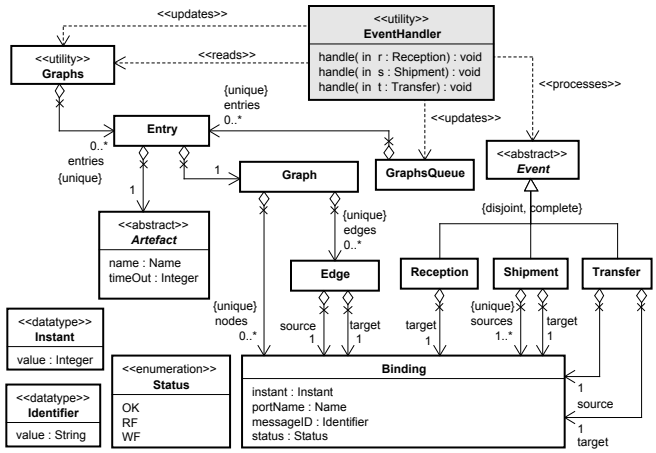


Figure 4. Model of the Event Handler module.

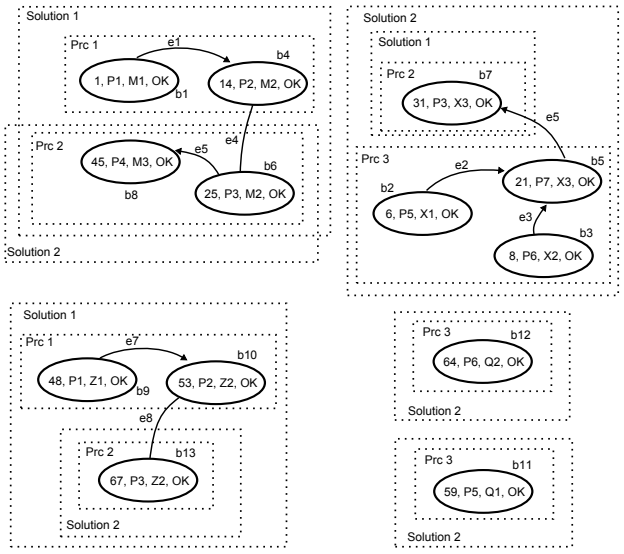


Figure 5. Sample Graphs Database.

Error Detector, three databases called Descriptions Database, Graphs Database, and History Database, and a queue called Graphs Queue.

The Registrar module is responsible for maintaining the Descriptions Database up to date. This database provides the other modules a description of the solutions, processes, ports, and rules the monitor handles. Figure §3 presents the abstract model of this module. (Note that we use term 'artefact' to refer to both solutions and processes.)

The Event Handler uses the events reported by ports to update the Graphs Database and the Graphs Queue. Figure §4 presents an abstract model of this module. An event can be of type Reception, which happens at ports that read data from an application (either successfully or unsuccessfully) and other ports that fail to read data at all, Shipment, which occurs when a port writes information (either successfully or unsuccessfully), and Transfer, which happens when a port succeeds to read data that was written previously by another port. Every event has a target binding and zero, one, or more source bindings. We use this term to refer to the data involved in an event, namely: the instant when the event happened, the name of the port, the identifier of the message read or written, and a status, which can be either OK to mean that no problem was detected, RF to mean that there was a reading failure, or WF to mean that there was a writing failure.

The Graphs Database stores an entry per artefact in the Descriptions Database; such entries contain a graph that the Event Handler builds incrementally, as it receives events. Figure §5 shows a sample Graphs Database for the EAI solution in Figure §1. For instance, let us focus on bindings b_6 and b_4: the former is involved in process Prc_2 and both solutions, and it denotes that port P_3 dealt with message M_2 at instant 25, and that the result was successful; the later is involved in process Prc_1 and $\mathsf{Solution}_1$ only, and it indicates that port P_2 dealt with message M_2 at instant 14, and that the result was successful; furthermore, the edge between them both indicates that binding b_6 originates from binding b_4.

The Graphs Queue is used to refer to the entries in the Graphs Database that have changed since the database was analysed for the last time. This helps minimise the work performed by the Error Detector, whose abstract model is presented in Figure §6. Note that it is relatively easy to find correlations in a graph like the one in Figure §5 since this task amounts to finding the connected components of the graph [9]. Contrarily, verifying them depends completely on the semantics of the EAI solutions involved. This is why we assign each artefact an upper bound to the total amount of time it is expected to take to produce a valid correlation, i.e., a time out, and a set of rules of the following form, cf. Figure §3:

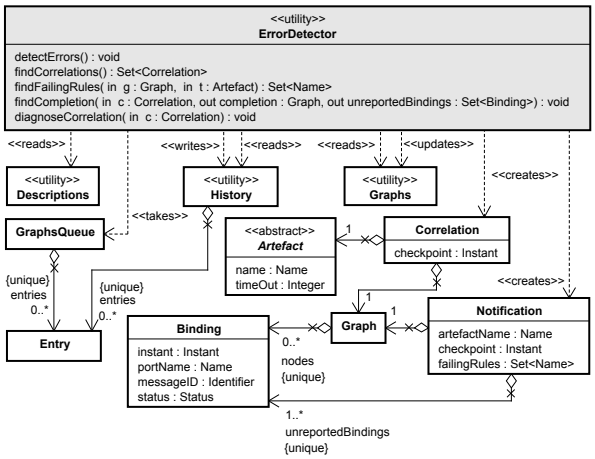


Figure 6. Model of the Error Detector module.

$\mathsf{P}_1[m_1..n_1], \ldots, \mathsf{P}_p[m_p..n_p] \rightarrow \mathsf{Q}_1[r_1..s_1], \ldots, \mathsf{Q}_q[r_q..s_q]$,

where P_i and Q_j are port names and m_i, n_i, r_i, s_i denote the minimum and maximum number of messages a correlation allows in each port so that it can be considered valid. For instance, a rule like $\mathsf{P}_5[1..1], \mathsf{P}_6[1..1] \rightarrow \mathsf{P}_4[1..10]$ regarding $\mathsf{Solution}_2$ in Figure §1 means that it is a requirement for a correlation to be considered valid that it has one message at port P_5, one message at port P_6 and then 1–10 messages at port P_4.

The correlations found by the Error Detector module are removed from the Graphs Database and stored in the History Database. This helps us complete them if new messages are reported later.

IV. Detecting Errors

Due to space limitations, we do not provide any additional details on the Registrar or the Event Handler modules. Instead, we focus on the Error Detector, which is the central module. Its algorithm is as follows:

```
1: to detectErrors() do
2:     repeat
3:         s = findCorrelations()
4:         for each correlation c in s do
5:             verifyCorrelation(c)
6:         end for
7:     end repeat
8: end
```

It runs continuously; in each iteration, it first finds a set of correlations and then verifies them sequentially. In the following subsections, we delve into the algorithms to find correlations and to verify them.

A. Finding Correlations

The algorithm to find correlations is as follows:

```
1: to findCorrelations(): Set(Correlation) do
2:     take entry f from the Graphs Queue
3:     checkpoint = getTime()
4:     s = find connected components of f.graph
5:     result = ∅
6:     for each graph g in s do
7:         c = new Correlation(artefact = f.artefact, graph = g,
8:                             checkpoint = checkpoint)
9:         add c to result
10:    end for
11: end
```

This algorithm starts by taking an entry f from the Graphs Queue at line §2; if there is not an entry available, then we assume that the algorithm blocks here until an entry is available. Note that the core of the algorithm is line §4, in which we find the connected components of the graphs that corresponds to the entry we have taken from the Graphs Queue. We rely on the well-known Hopcroft and Tarjan's algorithm to find connected components since it runs in linear time and has no additional space requirements [9]. In the loop at lines §6–§10, we simply use the components we have found to create the corresponding Correlation objects.

B. Verifying Correlations

A correlation can be diagnosed as on-going, valid or invalid. A correlation is on-going if its deadline has not expired yet. Bear in mind that correlations are analysed within the context of an artefact, which must have an associated time out and set of rules. The deadline for a correlation is defined as the time of its earliest binding plus this time out. This provides a time frame within which all of the messages involved in the correlation are expected to be reported. A correlation is valid if all of the messages it involves were read or written by the expected deadline, there was no reading or writing failure, and all of the rules involved are passed; otherwise, it is considered invalid and a notification must be generated so that it can be diagnosed and appropriate recovery actions can be executed later.

The algorithm to verify a correlation is as follows:

```
1: to verifyCorrelation(in c: Correlation) do
2:     findCompletion(c, out completedGraph, out unnotifiedBindings)
3:     status = every binding b in completedGraph.nodes has status OK?
4:     earliestInstant = minimum of completedGraph.nodes.instant
5:     latestInstant = maximum of completedGraph.nodes.instant
6:     deadline = earliestInstant + c.artefact.timeOut
7:     notPassedRules = checkRules(completedGraph, c.artefact)
8:     isValid  = deadline <= c.checkpoint and latestInstant <= deadline and
9:                status == true and notPassedRules == ∅
10:    isInvalid = (deadline < latestInstant) or
11:                (deadline <= c.checkpoint and (not status or notPassedRules ≠ ∅))
12:    if isValid then
13:        f = find the entry for c.artefact in the Graphs Database
14:        remove c.graph from f.graph
15:        g = new Entry(artefact = c.artefact, graph = c.graph, isValid = true)
16:        add g to History database
17:    elsif isInvalid then
18:        f = find the entry for c.artefact in the Graphs Database
19:        remove c.graph from f.graph
20:        g = new Entry(artefact = c.artefact, graph = completedGraph,
21:                      isValid = false)
22:        add g to History database
23:        n = new Notification( artefactName = c.artefact.name,
```

```
24:                     graph = completedGraph,
25:                     unnotifiedBindings = unnotifiedBindings,
26:                     checkpoint = c.checkpoint,
27:                     notPassedRules = notPassedRules)
28:       send n to the notification port of the monitor
29:    elsif
30:       - Nothing to do, since c is an on-going correlation
31:    end if
32: end
```

The algorithm gets a Correlation c as input; the first thing it has to do is to complete it with the help of the History Database. Note that correlations that are not on-going are removed from the Graphs Database; due to the asynchronous nature of EAI solutions, that implies that a after a correlation is verified, additional correlated messages may be reported. This is the reason why before verifying a correlation, it must be completed using the History Database. Algorithm findCompletion, which is explained later, performs this tasks; given a correlation c, it returns a graph that includes $c.graph$ and additional nodes and edges found in the History Database, as well the subset of bindings in the completed correlation that have not been notified, yet. In lines §3–§10 it calculates the status of the correlation, its deadline, the set of rules that are not passed, and determines if the correlation is valid, invalid or on-going. (Note that a correlation is on-going when it is neither valid nor invalid.) If correlation c is found to be valid, we then locate the entry that corresponds to its associated artefact in the Graphs Database, remove the correlation from it, and create a new entry in the History database (lines §13–§16). If it is found to be invalid, the process is similar, but a Notification object is created and sent to the notification port of the monitor so that the correlation can be diagnosed and the appropriate recovery actions can be executed. If it is an on-going correlation, then we just have to wait.

C. Completing Correlations

The algorithm to complete a correlation is as follows:

```
 1: to findCompletion(in c: Correlation,
 2:                   out completedGraph: Graph,
 3:                   out unnotifiedBindings: Set(Binding) ) do
 4:    completedGraph = new Graph(nodes = shallow copy of c.graph.nodes,
 5:                               edges = shallow copy of c.graph.edges)
 6:    unnotifiedBindings = shallow copy of c.graph.nodes
 7:    s = find all of the entries for c.artefact in the History database
 8:    for each entry f in s do
 9:       intersection = c.graph.nodes ∩ f.graph.nodes
10:       if intersection ≠ ∅ then
11:          merge f.graph into completedGraph
12:          if not f.isValid then
13:             remove intersection from unnotifiedBindings
14:          end if
15:       end if
16:    end for
17: end
```

This algorithm takes a correlation c as input and returns a graph that is a completed version of $c.graph$ and a set of bindings that have not been notified so far. It first creates an initial completed graph at line §4 from a shallow copy of the nodes and the edges of the graph of correlation c. A shallow copy is made because otherwise line §11 would modify the original graph in correlation c. Line §6 also makes a shallow copy of all bindings from correlation c into the set of unnotified bindings, i.e., we initially assume that all of them have been notified. At line §7, the algorithm finds all entries for the artefact associated with correlation c and stores them in variable s. The loop at lines §8–§16 iterates over all of the entries in s; it discovers if there are common bindings between correlation c and entry f. This is done at line §9 by calculating the intersection amongst the bindings of c and the bindings of f. If the intersection returns a non-empty set, it means that the bindings of f can complete the bindings of c; in this case, the graph associated with entry f must be merged into the resulting completed graph at line §11. Line §13 removes the bindings that were detected to be already in the graph of entry f from the set of unnotified bindings, leaving only new bindings that were not reported yet. Note that this is done only if graph f represents an invalid graph; otherwise all bindings are new.

D. Checking Rules

The algorithm to check rules is as follows:

```
 1: to checkRules(in g: Graph, in t: Artefact): Set(Name) do
 2:    result = ∅
 3:    for each rule r in t.rules do
 4:       for each atom a in r.atoms do
 5:          n = count bindings b in g.nodes such that b.portName == a.port.name
 6:          if n < a.min or n > a.max then
 7:             add r.name to result
 8:          end if
 9:       end for
10:    end for
11: end
```

This algorithm takes a graph that represents a correlation and an artefact as input; it returns the subset of rules associated with the artefact that the correlation does not pass. The loop at lines §3–§10 iterates over the rules and the internal loop at lines §4–§9 checks every atom. The algorithm is simple since we just need to count the number of bindings that involve the port referenced in the atom; if this figure is not within the margins that the atom establishes, then it is added to the result of the algorithm since that rule is not passed.

V. Analysis of the Proposal

In this section, we analyse our proposal from a both a theoretical and a practical point of view. We first prove that it behaves linearly, i.e., it is is computationally tractable.

Theorem 1: Algorithm detectErrors() takes $O(b + c\,h)$ time to process an entry in the Graphs Queue, where b denotes the average number of bindings that have been reported since the last checkpoint, c denotes the average number of correlations found at each checkpoint, and h the average number of entries for an artefact in the History Database.

Proof: According to Theorems §2 and §4 in Appendix §A, lines §3 and §5 of the algorithm run in $O(b)$

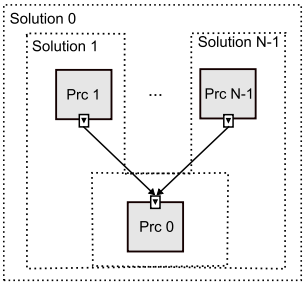
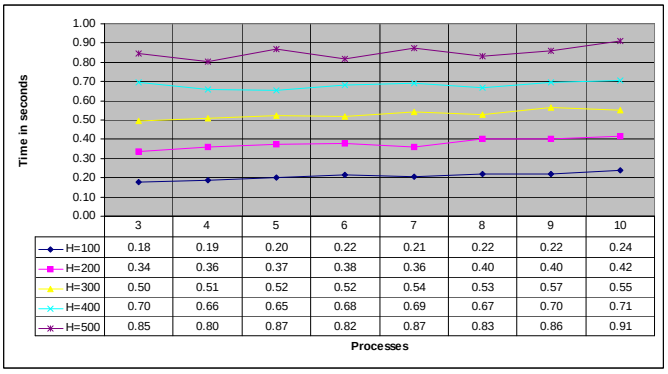


Figure 7. Experimental system.



	3	4	5	6	7	8	9	10
H=100	0.18	0.19	0.20	0.22	0.21	0.22	0.22	0.24
H=200	0.34	0.36	0.37	0.38	0.36	0.40	0.40	0.42
H=300	0.50	0.51	0.52	0.52	0.54	0.53	0.57	0.55
H=400	0.70	0.66	0.65	0.68	0.69	0.67	0.70	0.71
H=500	0.85	0.80	0.87	0.82	0.87	0.83	0.86	0.91

Figure 8. Experimental results.

and $O(h)$ time, respectively. Assume that findCorrelations() returns c correlations in average. Therefore, the loop at lines §4–§6 iterates c times in average. As a conclusion, algorithm detectErrors() takes $O(b + c\,h)$ time to process an entry in the Graphs Queue. ■

Note that there must be an upper bound to the average number of bindings reported between checkpoints; such bound is unknown since it depends on the hardware used, but it exists as long as the Descriptions Database does not change, i.e., no new artefacts are added to the system, and the message production ration is not monotonically increasing. The previous assumptions seem sensible since a typical company does no introduce new artefacts day after day and they cannot grow their hardware continusously. In turn, this implies that there must be an upper bound to the number of correlations that can be found in real-world scenarios. Contrarily, h increases monotonically as time goes by. This implies that after some time, the complexity of algorithm detectErrors() is dominated by h, i.e., the algorithm behaves linearly in the average number of entries per artefact in the History Database.

To prove that our algorithm makes sense in practice, we have also carried out a series of experiments. Due to space limitations, we report on one of the worst-case scenarios only, cf. Figure §7. It consists of $N-1$ processes that report to a single process denoted as Prc_0. Note that there are N solutions and that process Prc_0 belongs to each of them; consequently, every time a message is sent to this process, $N+2$ artefacts are involved (Prc_0, the process that sent the message, and the solutions).

The experiments were run on a machine that was equipped with an Intel Pentium D processors that ran at 3.4Ghz, had 2 GB of RAM memory, Windows Server 2003 (32-bit edition), and JRE 1.6. Each experiment consisted of executing the previous system for 24 hours with a fixed number of processes and a fixed maximum history size; these parameters changed from experiment to experiment. We set the fault rate at 10%, i.e., one out of every 10 messages was not delivered successfully, was intentionally replicated, or contained erroneous data with equal probability. Note that other parameters equal, the fault rate does not have an impact on the efficiency of our algorithm, since checking a correlation takes $O(r\,a)$ time, where r denotes the number of rules associated with an artefact and a the average number of atoms in these rules, cf. Theorem §5 in Appendix §A. The time to transmit messages was less than a millisecond since we considered small-sized messages that were sent across a high-speed local area network. The message production rate was set to a message per second to simulate a continuously-loaded system.

Figure §8 shows our results; the abscissa reports on the number of processes in the system, which varied from $P = 3$ to $P = 10$, and the ordinate reports on the average time the algorithm to detect errors took; each line represents the results we gathered when we varied the history size from $H = 100$ to $H = 500$ in steps of 100. The conclusion is that adding new processes to the system has obviously an impact on the time to detect errors, since these processes result in additional bindings that need to be processed by our algorithm. The impact is however linear, with a slight slope. In average, the impact of adding a new process to the system is 0.022 ± 0.017 seconds. Increasing the maximum size of the History Database by 100 entries also has an impact of 0.161 ± 0.022 seconds in average.

VI. Conclusions and Future Work

We have presented a proposal to detect errors in the context of EAI solutions. It is novel in that it is not bound to orchestrations or choreographies, neither to a process- nor a task-based execution model; it is totally independent. We have also proven that its time complexity is $O(b + c\,h)$, where b denotes the average number of bindings that have been reported by means of events since the last checkpoint, c denotes the average number of correlations found at each checkpoint, and h the average number of entries for an artefact in the History Database. Recall that the only purpose of this database is to complete correlations that are found in the Graphs Database, just in case a message is processed by a port after the deadline for the corresponding correlation expires. In practice, it makes sense to remove old information from the database periodically; this puts an upper bound to the size of the History Database, which, in turn, puts an upper bound to the total time the algorithm may take to detect errors. The experimental results prove that not only is the proposal computationally tractable, but also efficient. Future work includes exploring how the proposal may benefit from multi-threading and reducing the amount of work required to analyse the database; note that the same

correlation may be analysed several times in cases in which solutions overlap.

APPENDIX

Theorem 2: Algorithm findCorrelations() terminates in $O(b)$ time, where b is the number of bindings that have been reported by means of events since the last checkpoint for a given artefact.

Proof: This algorithm takes an entry from the Graphs Queue whenever it is available. Note that this may take an arbitrary time since it depends on the events being handled, which, in turn, depends on the system being monitored. Therefore, the time complexity refers to the time the algorithm takes to process an entry once it is taken from the Graphs Queue. The instruction at line §3 runs in $O(1)$ time; contrarily, the instruction at line §4 has to find the connected components of the graph associated with an entry, which is accomplished in $O(\max\{b, r\})$ time [9], where b denotes the number of nodes in the graph being analysed (bindings), and r the number of edges amongst them. In our scenario, it is expected that $b \approx r$ since edges are added to a graph when a shipment event is handled (n source bindings, one target binding, then n edges), or when a transfer event is handled (one source binding, one target binding, then one edge). Thus, without loss of generality, we can assume that line §4 runs on $O(b)$ time. The loop at lines §6–§10 iterates over each connected component to create new correlations, i.e., it runs in $O(c)$ time, where c denotes the average number of connected components found. As a conclusion, findCorrelations() terminates in $O(b + c)$ time. Note that b is usually expected to dominate c, since the total number of bindings reported in between checkpoints is proportional to the number of correlations c, i.e., $b = k\,c$ for an unknown k. Thus, we can conclude that findCorrelations() actually runs in $O(b)$ time. ■

Theorem 3: Algorithm verifyCorrelation(c) terminates in $O(h)$ time, where h denotes the number of entries in the History Database that involve $c.artefact$.

Proof: This algorithm is dominated by the calls to algorithms findCompletion and checkRules at lines §2 and §7, respectively. The rest of the lines may be assumed to execute in $O(1)$ time since they involve iterating over a completion of a correlation, or manipulating their associated graphs, which is expected to involve a relatively small number of bindings. According to Theorems §2 and §4, lines §2 and §7 are expected to run in $O(h+r\,a)$ time, where h denotes the number of entries in the History Database that involve $c.artefact$, r is the number of rules associated with this artefact, and a is the average number of atoms in these rules. Note that this is expected to be dominated by $O(h)$ as time goes by and the History Database grows. Therefore, verifyCorrelation(c) terminates in $O(h)$ time. ■

Theorem 4: Algorithm findCompletion(c, out cg, out ub) terminates in $O(h)$ time, where h denotes the number of entries for artefact $c.artefact$ in the History Database.

Proof: The time complexity of this algorithm is dominated by the loop at lines §8–§16. It iterates over all of the entries associated with $c.artefact$ in the History Database. Let h denote the number of such entries. We can safely assume that the set operations within this loop can be implemented in $O(1)$ time, since they all operate on correlations, which are expected to involve a relatively small number of bindings. findCompletion(c, out cg, out ub) thus terminates in $O(h)$ time. ■

Theorem 5: Algorithm checkRules(g, t) terminates in $O(r\,a)$ time, where r denotes the number of rules associated with artefact t and a the average number of atoms in these rules.

Proof: The proof is straightforward since the loop at lines §3–§10 iterates a total of r times, where r denotes the number of rules associates with artefact t, and the loop at lines §4–§9 iterates a times in average, where a denotes the average number of atoms per rule. The algorithm then runs in $O(r\,a)$ time. ■

REFERENCES

[1] G. Alonso, C. Hagen, D. Divyakant, A. E. Abbadi, and C. Mohan. Enhancing the fault tolerance of workflow management systems. *IEEE Concurrency*, 8(3):74–81, 2000

[2] L. Baresi, S. Guinea, R. Kazhamiakin, and M. Pistore. An integrated approach for the run-time monitoring of BPEL orchestrations. In *ServiceWave*, pages 1–12, 2008

[3] M. Y. Chen, A. Accardi, E. Kiciman, D. A. Patterson, A. Fox, and E. A. Brewer. Path-based failure and evolution management. In *Networked Systems Design and Implementation*, pages 309–322, 2004

[4] D. K. W. Chiu, Q. Li, and K. Karlapalem. A meta modeling approach to workflow management systems supporting exception handling. *Inf. Syst.*, 24(2):159–184, 1999

[5] G. Dunphy and A. Metwally. *Pro BizTalk 2006*. Apress, 2006

[6] V. Ermagan, I. Krüger, and M. Menarini. A fault tolerance approach for enterprise applications. In *IEEE SCC*, pages 63–72, 2008

[7] J. B. Goodenough. Exception handling: Issues and a proposed notation. *Commun. ACM*, 18(12):683–696, 1975

[8] C. Hagen and G. Alonso. Exception handling in workflow management systems. *IEEE Trans. Software Eng.*, 26(10):943–958, 2000

[9] J. E. Hopcroft and R. E. Tarjan. Efficient algorithms for graph manipulation [h] (algorithm 447). *Commun. ACM*, 16(6):372–378, 1973

[10] C. Ibsen and J. Anstey. *Camel in Action*. Manning Publications, 2010

[11] L. Li, C. N. Hadjicostis, and R. S. Sreenivas. Designs of bisimilar petri net controllers with fault tolerance capabilities. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 38(1):207–217, 2008

[12] C. Liu, M. E. Orlowska, X. Lin, and X. Zhou. Improving backward recovery in workflow systems. In *Database Systems for Advanced Applications*, pages 276–286, 2001

[13] M. Sampath, R. Sengupta, and S. Lafortune. Failure diagnosis using discrete-event models. *IEEE Trans. on Control Syst. Technol.*, 4(2):105–124, 1996

[14] Y. Yan, M.-O. Cordier, Y. Pencolé, and A. Grastien. MonitoringWeb service networks in a model-based approach. In *International Conference on Web Services*, pages 192–203, 2005

[15] Y. Yan and P. Dague. Modeling and diagnosing OrchestratedWeb service processes. In *International Conference on Web Services*, pages 51–59, 2007

Feature Model Validation: A Constraint Propagation-Based Approach

Guoheng Zhang, Huilin Ye, and Yuqing Lin
School of Electrical Engineering and Computer Science
University of Newcastle
Callaghan 2308, NSW, Australia

Abstract - *Feature model validation aims to identify errors in feature models. The two major errors, called dead features and false variable features, are caused by contradictory feature relationships in a feature model. Current existing approaches use constraint satisfaction problem (CSP) and CSP solvers to identify these feature model errors. However, CSP is a NP-complete problem and CSP solvers reveal a weak time performance. To overcome this limitation, we develop a constraint propagation based approach to identify dead features and false variable features. The correctness and efficiency of our approach is compared with a well known feature model validation tool, called FAMA, based on a number of large-size feature models which are randomly generated. The time spent for identifying feature model errors is significantly reduced from O (2^n) required by FAMA which uses CSP solvers to O (n^2), while both approaches identified the same set of errors for all the evaluated feature models.*

Keywords: feature model validation, dead features, false variable features, constraint propagation, feature models.

1 Introduction

Currently, feature models are widely used to represent the commonalities and variabilities of software product line (SPL) members [1]. A feature model consists of features and feature relationships among the features. From a feature model, a specific product can be derived during product configuration which is the process of selecting the desired features from a feature model based on customer' requirements and feature relationships specified in a feature model.

In a feature model, there may exist contradictory feature relationships. For example, feature *A* requires feature *B* and feature *B* excludes with feature *A*. Obviously there are contradictory feature relationships between *A* and *B*. The contradictory feature relationships will result in feature model errors, such as dead features and false variable features, which prevent a feature model from representing the right set of product line members desired by the domain. Therefore, these feature model errors must be detected and corrected for an effective product configuration. It is a difficult and time-consuming task to identify these feature model errors for large-size feature models. Therefore, an automated method of identifying feature model errors has been recognized as one of challenges in the area of feature model validation [2].

Several feature model errors have been defined in literature, such as dead features, false variable features, redundancies, wrong cardinality and void feature model [3]. Among these feature model errors, dead features and false variable features have been recognized as the most critical errors [3-6]. A variable feature is dead if it cannot appear in any software product line (SPL) member and a variable feature is false variable if it must be included into all SPL members [5]. In this paper, we concentrate on the identification of dead features and false variable features.

The most promising approach of identifying feature model errors is proposed by *Trinidad* [5] and *Czarnecki* [7]. This approach is based on constraint satisfaction problem (*CSP*) which transforms features into variables and feature dependencies into constraints over the variables. To check whether a feature is dead, this approach assigns the checked feature with value "true" and then uses *CSP* solvers to find valid solutions over the other variables based on the constraints. If no valid solutions can be found, the checked feature is dead. The false variable features are identified in a similar way except that the checked feature is assigned with value "false". Although this approach identifies feature model errors automatically, it is quite inefficient, because *CSP* is a NP-complete problem and the number of solutions that need to be examined is 2^n where n is the number of features in a feature model. If the number of features in a feature model is large, the *CSP*-based approach is inefficient to identify feature model errors.

In this paper, we propose a recursive algorithm to identify dead features and false variable features based on constraint propagation, which is the process of propagating the inclusion or removal of a variable feature to a set of other features through different feature relationships. A set of constraint propagation rules are proposed. The time spent for identifying feature model errors is significantly reduced from $O(2^n)$ in [5] to $O(n^2)$.

The rest of this paper is organized as follows: section 2 will introduce some background knowledge on feature models and feature model errors. In section 3, we will introduce constraint propagation and propose the formalized constraint propagation rules. In section 4, we develop a recursive algorithm to identify feature model errors based on constraint propagation rules. In section 5, we evaluate the correctness and efficiency of our approach by comparing with a well known feature model validation tool, called *FAMA*, based on a

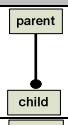
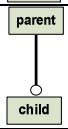
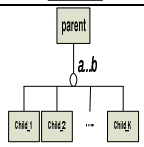
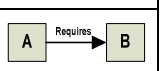
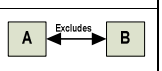
number of randomly generated feature models. Finally we conclude this paper and identify the future work in Section 6.

2 Background

2.1 Feature Models

A feature model represents the commonalities and variabilities of member products in a software product line (SPL) in terms of features. A feature model is mostly represented as a feature tree where nodes represent features and edges represent relationships among features. Since Kang et al., [8] first introduced the basic feature model in FODA several extensions have been proposed, such as feature attributes, feature cardinality and group cardinality [9-11]. Cardinality-based feature model (*CBFM*) which extends the basic feature model with group cardinality, is most widely used because of its conceptual completeness. There are two kinds of feature relationships in a *CBFM*: hierarchical relationship and cross-tree feature dependency. A hierarchical relationship describes the selection relationship between a parent feature and its child features, including mandatory, optional and feature group with group cardinality. A feature dependency describes feature selection constraint between two cross-tree features, including requires and excludes. In this paper, we adapt cardinality-based notations to represent feature models. The semantics and notations of feature relationships in a *CBFM* are shown in table 1.

Table 1 The semantics and notations of feature relationships in *CBFM*

Feature Relationship	Semantics	Notation
Mandatory	if the parent feature is selected, the child feature which has a mandatory relationship with its parent must be selected.	
Optional	if the parent feature is selected, the child feature which has an optional relationship with its parent may or may not be selected.	
Feature Group	if the parent feature is selected, at least *a* features must be selected and at most *b* features can be selected from the feature group based on the group cardinality [*a*…*b*].	
Requires	"A requires B" means that if feature A is selected, feature B must be selected.	
Excludes	"A excludes B" means that A and B cannot be selected into the same member product.	

A feature model can represent all potential member products of a SPL in terms of features and feature relationships among features. A specific member product can be derived from a SPL in feature-based product configuration during which the desired features are selected based on customers' requirements. The feature selection is also constrained by all the feature relationships specified in a feature model. Fig. 1 shows a *CBFM* of a simplified tourist guide SPL.

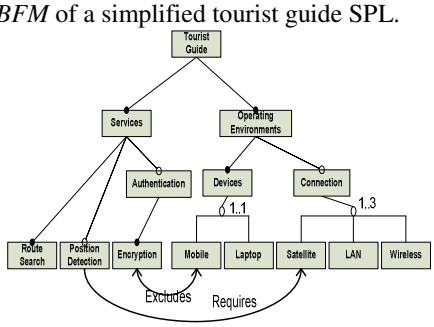


Figure 1 A cardinality based feature model of tourist guide SPL

2.2 Feature Model Errors

A feature model is designed to represent a software product line (SPL) and should include all potential member products of the SPL. Feature modelers use feature relationships to constrain the set of products that can be derived from a feature model. However, there may exist contradictory feature relationships which will cause feature model errors, such as dead features, false variable features and wrong cardinalities. These feature model errors prevent a feature model from representing the right set of products desired by the SPL domain. *Trinidad* et al. [5] has defined the two most critical feature model errors as follows:

- Dead features. A dead feature is a non-instantiable feature, i.e. a feature that despite of being defined in a feature model, it appears in no product in software product line.

- False variable features. A child feature in a non-mandatory relationship is a false variable feature if it has to be instantiated whenever its parent feature is selected.

Based on the above definition of dead features and false variable features, to check whether a feature *f* is a dead feature, we assume that *f* is not dead and include *f* into a product. When *f* is included, its neighbor features, such as parent, child, dependent features, will be included or excluded based on their specific relationship with *f*. Similarly, the inclusion or exclusion of the neighbor features will result in inclusions or exclusions of their neighbor features as well, and so forth. This transmission process through different feature relationships is called feature relationship propagation. If the inclusion of *f* will lead to a contradictory conclusion of exclusion of *f* through feature relationship propagation, the original assumption that *f* is not a dead feature is wrong. Then we can conclude that *f* is a dead feature. In a similar way, to check whether a feature *f* is a false variable feature, we assume that *f* is not false variable and remove *f* from products. If the exclusion of *f* will lead to a contradictory conclusion of inclusion of *f* through feature relationship propagation, the

original assumption that *f* is not a false variable feature is wrong. We can then conclude that *f* is a false variable feature.

In this paper, we concentrate on identifying dead features and false variable features through the feature relationship propagation. As feature relationships constrain the selection of features in product configuration, feature relationship propagation can also be called constraint propagation. In this paper, an approach, called constraint propagation method, is developed to identify feature model errors, including dead features and false variable features. Details of this approach will be described in the next two sections.

3 Constraint Propagation

Constraint propagation is the process of propagating the decision on a variable feature, either removal or inclusion, to a set of other features in the feature model through different feature relationships, including hierarchical relationships and cross-tree feature dependencies. The constraint propagation is an iterative process. At the first propagation step, the inclusion or removal of a variable feature will be propagated to its neighbor features which are closely connected with the variable feature by one single feature relationship. Then each newly removed or included feature in the last propagation step will propagate the decision on it to its neighbor features iteratively. The propagation process will stop until no more features can be removed or included. The key issue of constraint propagation is the propagation rules about how to propagate the decision on a feature to its neighbor features. To propose the propagation rules, we first give some definitions that are used to explain the propagation rules.

Definition 1: (feature attribute: status): Every feature has an attribute named as "status" that ranges over the domain {*-1*, *0*, *1*}. The attribute "status" gets the value "*1*" if the feature is included to be a part of the product, gets the value "*-1*" if the feature is removed from the product and otherwise "*0*".

Definition 2: (action: include or remove): In a product configuration, the action of changing the status of a feature *f* from "*0*" to "*1*" is named as "include" and represented as *INclude (f)* while the action of changing the status of a feature *f* from "*0*" to "*-1*" is named as "remove" and represented as *REmove (f)*.

Definition 3: In a feature group $FG = \{f_1, f_2 \ldots f_n\}$, the number of features whose statuses are "*1*" is represented as *NumberofINcluded (FG)* while the number of features whose statuses are "*-1*" is represented as *NumberofREmoved (FG)*. The number of features in *FG* is represented as *NumberofFeatures (FG).*

Definition 4: (neighbor features): The neighbor features of a certain feature *f* include all the features that connect with *f* by a single feature relationship, including mandatory, optional, feature group with cardinality, requires or excludes. We represent the neighbor features of a certain feature *f* using a set of notations as follows:

- Parent Feature*: f* has a parent feature if *f* is not root feature of a feature model. We use *f. parent* to represent the parent feature of *f*. If *f* is a root feature, *f. parent* has the value "*null*". We use *f. pr* to represent the hierarchical relationship the connects feature *f* with its parent feature:
  - *f. pr = m*, if a mandatory relationship connects feature *f* with its parent feature.
  - *f. pr = o*, if an optional relationship connects feature *f* with its parent feature.
  - *f. pr = c*, if a feature group with cardinality connects feature *f* with its parent feature.
- Child Feature: *f* has a number of child features if *f* is not leaf feature of a feature model. We use *f. set (child)* to represent the set of child features of feature *f*. The feature set *f. set (child)* can be decomposed into several subsets based on the hierarchical relationships that connect *f* with its child features: $f.\ set\ (child) = f.\ set\ (m) \cup f.\ set\ (o) \cup f.\ set\ (c_1) \ldots \cup f.\ set\ (c_n)$.
  - The notation *f. set (m)* illustrates the set of child features which connect with *f* by mandatory relationships.
  - The notation *f. set (o)* illustrates the set of child features which connect with *f* by optional relationships.
  - The notation $f.\ set\ (c_i)$ illustrates a feature group under *f*. Feature *f* may correspond to a number of feature groups and we use $\{f.\ set\ (c_1), f.\ set\ (c_2) \ldots f.\ set\ (c_n)\}$ $(n>=0)$ to represent the feature groups under feature *f*.
- Brother Features: *f* has a set of brother features if *f* belongs to a feature group. We use the notation *f. set (brother)* to represent the set of features which belong to the same feature group with *f*.
- Friend Features: *f* has a set of friend features if *f* connects with other features by requires or excludes. We use the notation *f. set (friend)* to represent the set of features which connect with *f* by cross-tree feature dependencies. The feature set *f. set (friend)* can be decomposed into three subsets based on different feature dependency types that connect *f* with its friend features: $f.\ set\ (friend) = f.\ set\ (reqed) \cup f.\ set\ (reqs) \cup f.\ set\ (exc)$.
  - The notation *f. set (exc)* represents the set of features that *f* excludes.
  - The notation *f. set (reqed)* represents the set of features that require *f*.
  - The notation *f. set (reqs)* represents the set of features that *f* requires.

Based on the above notations, we can represent the neighbor features for any feature in a feature model. In table 2, we take two features "devices" and "satellite" in the tourist guide feature model of Fig. 1 to illustrate how to represent the neighbor features of a feature in a feature model.

Table 2 The neighbour features of "devices" and "satellite" in tourist guide SPL feature model of Fig. 1.

feature *f*	devices	satellite
f. parent	operating environment	connection
f. pr	m	c
f. set (m)	∅	∅
f. set (o)	∅	∅
f. set (c_1)	mobile, laptop	∅
f. set (brother)	∅	LAN, wireless
f. set (exc)	∅	∅
f. set (reqed)	∅	position detection
f. set (reqs)	∅	∅

The action on a certain feature has different impacts to its neighbor features based on the semantics of different feature relationships connecting the certain feature and its neighbor features. Based on the semantics of feature relationships in table 1, the detailed propagation rules from a variable feature *f* to its different kinds of neighbor features are described as follows:

INclude (f): when *f* is included into products, the neighbor features of *f* will be affected as follows:

- Parent feature: when *f* is not the root feature, if *f* is included, its parent feature will be included. The propagation rule is:

$$1.\quad (f.parent != null) \Rightarrow INclude(f.parent)$$

- Child feature: when *f* is not leaf feature and there are a set of child features *f. set (m)* that connect with *f* by mandatory relationships, if *f* is included into products, all the features in *f. set (m)* will be included. When there is a feature group *f. set (c_i)* with group cardinality [*a…b*] under *f*, if *f* is included and the number of removed features in *f. set (c_i)* reaches its maximum value (*NumberofFeatures (f. set (c_i))* – *a*), all features that are not determined in *f. set (ci)* will be included. The two propagation rules are:

$$2.\quad (f.set(m) != \varnothing) \Rightarrow \forall f_i \in f.set(m) \mid INclude(f_i)$$

$$3.\quad NumberofFeatures(f.set(ci)) - a = NumberofREmoved(f.set(ci)) \Rightarrow \forall f_i \in f.set(ci) \wedge f_i.status = 0 \mid INclude(f_i)$$

- Brother feature: when *f* is in a feature group with group cardinality [*a…b*] and has a set of brother features *f. set (brother)*, if *f* is included and the number of included features in *f. set (brother)* reaches its maximum value (*b-1*), all the features in *f. set (brother)* that are not determined will be removed. The propagation rule is:

$$4.\quad NumberofINcluded(f.set(brother)) = b-1 \Rightarrow \forall f_i \in f.set(brother) \wedge f_i.status = 0 \mid REmove(f_i)$$

- Friend feature: when there exists a set of features that *f* requires and a set of features that *f* excludes, if *f* is included, all features in *f. set (reqs)* will be included and all the features in *f. set (exc)* will be removed. The propagation rules are:

$$5.\quad f.set(reqs) != \varnothing \Rightarrow \forall f_i \in f.set(reqs) \mid INclude(f_i)$$

$$6.\quad f.set(exc) != \varnothing \Rightarrow \forall f_i \in f.set(exc) \mid REmove(f_i)$$

REmove (f): when *f* is removed from products, the neighbor features of *f* will be affected as follows:

- Parent feature: when *f* is not the root feature and *f* connects with its parent feature by mandatory relationship, if *f* is removed, its parent feature will be removed. When *f* is in a feature group with group cardinality [*a…b*] and has a set of brother features *f. set (brother)*, if *f* is removed and the number of removed features in *f. set (brother)* exceeds its maximum value *NumberofFeatures* (*f. set (brother)*)-*a*, the parent feature of *f* will be removed. The propagation rules are:

$$7.\quad (f.parent != null) \wedge (f.pr = m) \Rightarrow REmove(f.parent)$$

$$8.\quad (f.parent.status = 0) \wedge (NumberofREmoved(f.set(brother)) > NumberofFeatures(f.set(brother)) - a) \Rightarrow REmove(f.parent)$$

- Child feature: when *f* is not leaf feature and it has a set of child features *f. set (child)*, if *f* is removed, all the features in *f. set (child)* will be removed. The propagation rule is:

$$9.\quad f.set(child) != \varnothing \Rightarrow \forall f_i \in f.set(child) \mid REmove(f_i)$$

- Brother feature: when *f* is in a feature group with group cardinality [*a…b*] and has a set of brother features *f. set (brother)*, if the parent feature of *f* is included and the number of removed features in *f. set (brother)* reaches its maximum value, all the features in *f. set (brother)* that are not determined will be included. The propagation rules is:

$$10.\quad (f.parent.status = 1) \wedge (NumberofREmoved(f.set(brother)) = NumberofFeatures(f.set(brother)) - a) \Rightarrow \forall f_i \in f.set(brother) \wedge f_i.status = 0 \mid INclude(f_i)$$

- Friend feature: when there exists a set of features *f. set (reqed)* that requires *f*, if *f* is removed, all the features in *f. set (reqed)* will be removed. The propagation rule is:

$$11.\quad f.set(reqed) != \varnothing \Rightarrow \forall f_i \in f.set(reqed) \mid REmove(f_i)$$

4 Identifying Feature Model Errors

In this section, we develop a recursive algorithm to identify dead features and false variable features for a given feature model *FM* that is received as input in Algorithm 1. We traverse all the variable features (*FM. vars*) of feature model *FM*. For each variable feature *var*, we check whether the inclusion of *var* will result in contradictions through constraint propagation (line 3) and check whether the removal of *var* will result in contradictions through constraint propagation (line 9). Before identifying each error, the feature model will be initialized by function *REset* (*FM*) which sets the status of all full-mandatory features [12] as "1" and the status of all other features as "0". It should be noted that we set the status of the parent feature of *f* as "1" when we check whether *f* is a false variable feature based on the definition on false variable features in section 2.2.

Algorithm 1: Feature Model Error Identification

Parameters:

FM: the given feature model

Function **identifyErrors** (*FM*)

```
1.  for (each var in FM. vars) {
2.      REset (FM);
        //check whether var is a dead feature;
3.      if (propagate (var, 1)= cfs)
4.      printout "var is dead feature";
5.  end for
6.  for (each var in vars) {
7.      Reset (FM);
8.      var. parent. status = 1;
        //check whether var is a false variable feature;
9.      if (propagate (var, -1) = cfs)
10.     printout "var is false variable feature";
11. end for
```

The constraint propagation from a variable feature to other features is achieved through a recursive function "*propagate*" which takes two parameters: a feature *f* where the propagation starts in one propagation step and value *v* that is assigned to *f*. The "*v = 1*" illustrates the inclusion of *f* while "*v = -1*" illustrates the removal of *f*. The propagation process returns to the last recursive step in three conditions: if the status of *f* is different with *v*, a contradiction arises and we return to the last recursive step with a sign "*cfs*" which represents contradictory assignments (line 13); if the status of feature *f* is same with *v*, a overlapped assignment arises and we return to the last recursive step with a sign "*ols*" which represents overlapped assignments (line 16); if no contradictions arise after propagating value *v* of *f* to other features, we return to the last recursive step with a sign "*no propagation*" (line 33). The constraint propagation from *f* to its neighbor features is realized from line 19 to line 32. The function "*rule (fn, f, v)*" returns the value assigned to *fn* which is a neighbor feature of *f*, when *v* is assigned to *f* based on the propagation rules proposed in last section.

Function: Constraint Propagation

Parameters:

f: the feature where the propagation starts in one recursive step.

v: the value assigned to *f* based on the action taken on *f*: v = 1 when selecting *f* and v = -1 when removing *f*

Function **propagate** (*f, v*)

```
12. if (f. status = -v)          //contradictory assignments
13.     return cfs;
14. end if
15. if (f. status = v)
16.     return ols;              //overlapped assignments
17. end if
18. f. status = v                // assigning f with v
19. if (v = = 1)  // propagation from f when f is included
20.     for each neighbor feature fn of f
21.         if (propagate (fn, rule (fn, f, 1)) = = cfs)
22.             return cfs;
23.         end if
24.     end for
25. end if
26. if (v = = -1)     // propagation from f when f is removed
27.     for each neighbor feature fn of f
28.         if (propagate (fn, rule (fn, f, -1)) = = cfs)
29.             return cfs;
30.         end if
31.     end for
32. end if
33. return no-propagation;
```

5 Evaluation

In this section, we aim to evaluate our approach from two aspects: the correctness of the identified feature model errors and the efficiency for identifying feature model errors. We propose an evaluation process illustrated in Fig. 2. *FAMA* [13] which uses *CSP* solvers to identify and explain feature model errors is chosen as the contrast to evaluate our approach, as *FAMA* has been widely used in validating feature models and evaluated by many researchers. For a specific feature model, we use *FAMA* tool as well as our algorithm to identify feature model errors. The correctness of our approach can be verified by comparing the errors identified by our algorithm with the errors identified by *FAMA*. The efficiency of our approach can be assessed by comparing the time spent by *FAMA* with the time spent by our algorithm.

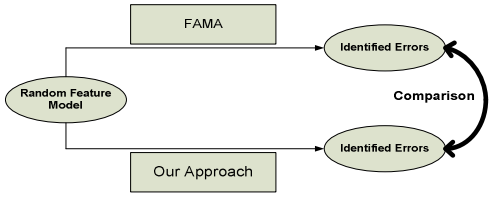


Figure 2 An evaluation process for correctness and efficiency of our approach

5.1 Experimental Platform

To perform our experiments, we develop our algorithm in Java, as *FAMA* is an Eclipse Project. The experiments were performed on a computer with an AMD 1.99 GHz CPU, 2 G of memory, Windows XP and 1.6 Java Virtual Machine (*JVM*). The maximum memory size of JVM is 512m (-Xmx512m) while the minimum memory size of JVM is 256m (-Xmx256m).

5.2 Random Feature Model

Feature model errors always exist in large-size feature models which have hundreds of thousands of features and feature relationships. Also, the advantage of our approach relies on the algorithm efficiency for identifying errors for large-size feature models. Therefore, the evaluation of our approach should be based on large-size feature models. It is difficult to obtain large-size feature model in real world SPL. To deal with this problem, we develop an algorithm which can generate random feature models. When generating the random feature models, we can set several parameters:

- the number of features in the feature model (*num_f*)
- the number of "requires" dependency in the feature model (*num_req*)
- the number of "excludes" dependency in the feature model (*num_exc*)
- the maximum number of mandatory child features of a certain feature (*max_mc*)
- the maximum number of optional child features of a certain feature (*max_oc*)
- the maximum number of feature groups under a feature (*max_gc*)
- the maximum number of features in a feature group (*max_var*)

Among these parameters, *num_f*, *num_req* and *num_exc* determine the size and complexity of a feature model while *max_mc*, *max_oc*, *max_gc* and *max_var* affect the depth and width of a fixed-size feature model. The values of *max_mc*, *max_oc*, *max_gc* and *max_var* should be set based on *num_f* to avoid that the feature model is too wide or too deep.

5.3 Correctness

The correctness of our approach heavily relies on whether the identified feature model errors by our algorithm are complete and correct. To evaluate the correctness of our approach, we generate 10 random feature models where the number of features ranges from 100 to 1000 and the number of dependencies ranges from 10 to 100. The parameters of the generation algorithm are set as follows: *max_mc*=3, *max_oc*=3, *max_gc*=3, *max_var*=4. Based on our experiments, when the number of features ranges from 100 to 1000, these parameters can satisfy that the feature model is neither too wide nor too deep.

Table 3 Dead features and false variable features detected by FAMA and our algorithm

Feature model	num_f	num_req	num_exc	FAMA		Our Algorithm	
				dead	false variable	dead	false variable
1	100	5	5	8	0	8	0
2	200	15	15	8	1	8	1
3	300	20	20	27	3	27	3
4	400	25	25	29	0	29	0
5	500	30	30	24	24	24	24
6	600	35	35	13	0	13	0
7	700	40	40	31	5	31	5
8	800	45	45	no result		56	3
9	900	50	50	no result		21	27
10	1000	50	50	no result		76	3

The number of dead features and false variable features identified by FAMA as well as our algorithm is shown in table 3. By comparison, the errors identified by our algorithm are the same as the errors detected by *FAMA* except for feature model 8, 9 and 10. For these three feature models, *FAMA* meets "out of heap memory" error and cannot identify feature model errors. However, the correctness of our approach can be verified by the other seven feature models in table 3.

5.4 Efficiency

The efficiency of our algorithm is reflected in identifying feature model errors for large-size feature models. In this experiment, we record the time spent by *FAMA* and our algorithm in identifying and explaining feature model errors for the ten feature models in table 3. The results are shown in table 4.

Table 4 The time spent for identifying feature model errors by FAMA and our approach

Feature model	num_f	num_req	num_exc	Our Algorithm (ms)	FAMA (ms)
1	100	10	10	15	1875
2	200	15	15	15	2203
3	300	20	20	143	19854
4	400	25	25	129	20345
5	500	30	30	986	125743
6	600	35	35	45	6534
7	700	40	40	1572	153983
8	800	45	45	1856	out_of_memory
9	900	50	50	1678	out_of_memory
10	1000	50	50	2145	out_of_memory

In table 4, the time unit is millisecond and from the result we can find that *FAMA* reveals a weak time performance when identifying feature model errors on medium and large size feature models and even meets "out of memory" error if the

number of features is larger. Compared with *FAMA*, our approach reveals a strong time performance. This is because *FAMA* adapts constraint programming to identify feature model errors and the number of solutions that need to be checked is 2^n where n is the number of features. Our approach significantly improves the efficiency because at most n^2 (n is the number of features) steps need to be traversed.

6 Conclusion and Future Work

In the area of feature model validation, a set of approaches have been proposed to identify feature model errors [2-7]. However, there is a critical limitation in the existing approaches. These approaches are inefficient because an exponential number of solutions need to be checked when identifying a feature model error for large-size feature models. To overcome this limitation, we develop a constraint propagation based algorithm to identify dead features and false variable features. The correctness and efficiency of our algorithm are evaluated based on a set of large size feature models which are randomly generated in our approach. In the future, we aim to detect the explanations for the identified errors in the constraint-propagation process.

7 Reference

[1] Kyo Chul Kang, "FODA: Twenty Years of Perspective on Feature Models," the keynote of SPLC 2009, San Francisco, CA, USA, 2009.

[2] D.Batory, D.Benavides and A.Ruiz-Cortes, "Automated Analysis of Feature Models: Challenges Ahead," in Communications of the ACM Vol. 49, No. 12. (2006) 45-47.

[3] David Benavides, Sergio Segura, Antonio Ruiz-Cortes, "Automated analysis of feature modes 20 years later: a literature review," in the journal of informaiton systems, 2010.

[4] Von der Massen, T., Lichter, H., " Deficiencies in feature models," in workshop on software variability management for product derivation-towards tool support.

[5] P. Trinidad *, D. Benavides, A. Duran, A. Ruiz-Cortes, M. Toro, "Automated error analysis for the agilization of feature modelling," in Journal of Systems and Software, 2007.

[6] A. Hemakumar, "Finding contradictions in feature models," in proceedings of the first international workshop on analysis of software product lines, 2008, pp. 183-190

[7] K. Czarnecki, S. Helsen and U. Eisenecker, "Formalizing cardinality-based feature mdoels and their specialization," in the journal of software process: improvement and practice 10 (1) (2005) 7-29.

[8] Kyo C. Kang, G. C. S., J. A. Hess, W.E. Novak and A. S. Petersem (1990). Feature-Oriented Domain Anaylisis (FODA) Feasibility Study. Technical Report CMU/SEI 90-TR-21, 1990.

[9] Czarnecki, K. Kim, P. , "Cardinality-based feature modelling and constraints: a progress report," in the proceeding of the International Workshop on Software Factories at OOPSLA 2005.

[10] David Benavides, Sergio Segura, Antonio Ruiz-Cortes, "Automated analysis of feature modes 20 years later: a literature review," in the journal of informaiton systems, 2010.

[11] M. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the RSEB," in Proceedings of the Fifth International Conference on Software Reuse, pages 76–85, Canada, 1998.

[12] H. Ye, Y. Lin, and W. Zhang, "Streamlined Feature Dependency Representation in Software Product Lines,"in international conference on software engineering research& practtice, Las Vegas, Nevada (2010)

[13] D, Benavides, S Segura, P Trinidad, A Ruiz-Cortés, "FAMA: Tooling a Framework for the Automated Analysis of Feature Models," in proceeding of the First International Workshop on Variability Modelling of Software-intensive Systems (VAMOS), 2007.

A Software Certification of Embedded Vehicle Platform Using Integrated Test

Hyun Chul Jo[1], Shiquan Piao[2], and Hui-Sup Cho[3]
Department Division of IT Convergence,
Daegu Gyeongbuk Institute of Science & Technology (DGIST), Daegu, Republic of Korea

Abstract - *This paper discusses various tests of the newest development tool for AUTOSAR based embedded platform. The growing number of electrics/electronics software in vehicle systems makes more and more necessary the increasing demands. For example, it needs the essential requirements such as ensuring reliability, low production cost, coping with limited resources, and so on. Recently, there have been relative studies that point to this issue. An AUTOSAR development partnership is such a case. AUTOSAR is a standardized automotive software architecture which is an alliance of OEM and supplier. Now, the focus is mainly directed at a source code generator that deals with the AUTOSAR standard concept. In this paper, we focus on new AUTOSAR RTE generator tool, through the integrated test for demanding vehicular applications. The result of this in-processing test satisfies a need for model requirements and standard methodology in an AUTOSAR specification.*

Keywords: Automotive Standard Software, Vehicle Platform, Source Code Development Tool, Software Testing, Integrated Test Platform

1 Introduction

Embedded software commonly has a significant role in automotive industries. In recent years, Electric/Electronic (E/E) software functionalities are increasingly given a great deal of weight in automotive systems. These increased uses of software emerge as a consequence of many defects. Besides, the expansion of embedded software emerges as a consequence of higher complexity and lower quality in automotive embedded control systems. Toyota recalled Prius, one of the first hybrid cards, in 2005. It began with a trivial programming error in its embedded software. However, the error finally stalled the gasoline engine.

Under these conditions, great attention has been shown to the question of Component Based software Development (CBD) researches. The concept is to increase the reusability of automotive platforms with a component unit, and it is necessary to catch up rapid rates of the change in software. An AUTomotive Open System ARchitecture (AUTOSAR) embedded software platform is such a trial. Automotive manufacturers and suppliers of Europe have planned out AUTOSAR projects.

This paper verifies the vehicular software development tool based on AUTOSAR standard. Before testing into model of the AUTOSASR RTE generator, we will revisit the functional and system design of the previous research.

In this paper, we first describe the AUTOSAR software platform and the concept of the RTE (Run Time Environment). Next, we introduce the pre-proposed the software development model which supports the AUTOSAR platform, specifically RTE generator and RTE template structure design. The software certification is done using the development and board testing of code warrior and free scale, respectively. Finally, we conclude and discuss future activities.

2 Automotive System and Software Engineering

2.1 AUTOSAR Conception

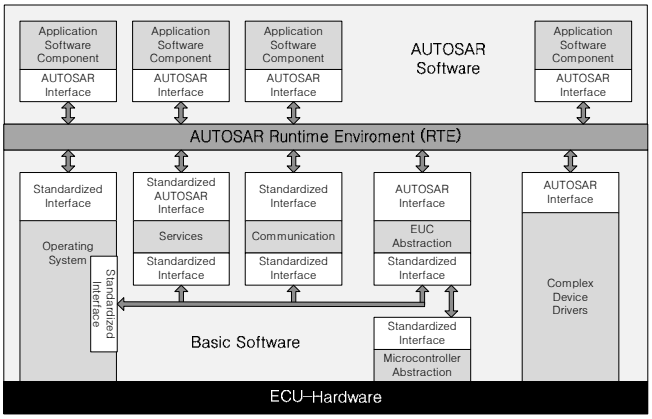


Figure 1. AUTOSAR ECU Software Architecture and Basic Software.

AUTOSAR provides a global standard for embedded automotive systems. Conceptually, it defines a modular software architecture. Each software component communicates via vehicle networks where a standardized interface is used to identify different software modules.

An AUTOSAR project was formally founded in June 2003 in which more than 100 company members were attended. Release 1.0 was published in 2005, and the specification of the Methodology and release 2.0 were introduced at the beginning of 2006. Through continuous revisions, the AUTOSAR specification has come to maturity more and more.

Fig. 1 shows the schematic view of the AUTOSAR ECU software architecture. The architecture is structured in 5 layers: Application layer, AUTOSAR Run-time Environment layer, Service layer (Included in OS, Services, Communication), ECU abstraction layer, and Microcontroller abstraction layer. All AUTOSAR software components including sensor and actuator components exist in the application layer. The software components in this layer communicate via RTE. The RTE layer implements AUTOSAR software components, which are independent of hardware. The service layer provides basic functions for ECU hardware, basic software modules, and applications. The ECU abstraction layer simply abstracts the ECU from the above layer. As illustrated, most of standardized interfaces are mapped into this layer. Lastly, the microcontroller abstraction layer is the lowest software layer that contains drivers.

2.2 Run Time Environment Concept

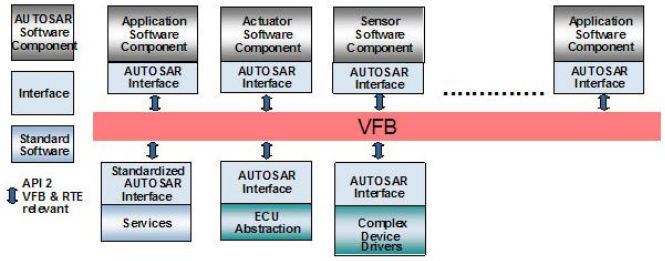


Figure 2. Overview about the Virtual Functional Bus.

Fig. 2 represents the basic concept of AUTOSAR RTE. Software Components (SW-C) conceptually mean application software that is independent of any ECUs and the location of the other software components. In SW-C descriptions, AUTOSAR provides each interface and other aspects. A Virtual Functional Bus (VFB) is provided as a mean to achieve the independence from hardware. It is the sum of all communication mechanisms, and it is a key signifying structure in the AUTOSAR software platform.

3 The Methodology of Design and Development

3.1 Functional and System Design

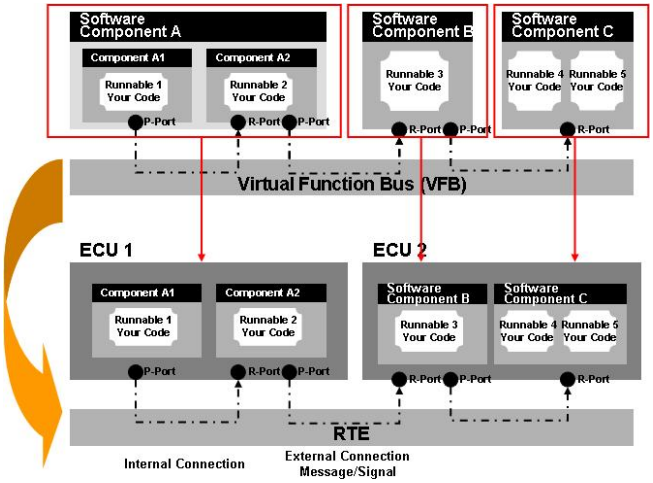


Figure 3. Basic Run Time Environment Approach.

The RTE works are classified as two things. First, it is an interface between software components and that enables it access to data outside. Second, it provides the infrastructure services that can communicate.

From this viewpoint, we can see that the Run-Time Environment (RTE) implements the VFB functionality. After that, software components are assigned to the each ECU in which this procedure is a data mapping. Before applying mapping steps, ECU and system constraint descriptions including ECU configure information are required to compose ECUs.

3.2 Model Development

The model development of RTE Generator is the previous in-depth research of the AUTOSAR platform. Conceptually, the RTE Generator means a automotive development tool which makes an application source code for vehicular software.

We designed the Generator to the XML parser, RTE Generator Engine (Generator init., Contract/Generation Module), RTE Template. The block diagram of this model is presented in Fig. 4.

An XML parser extracts systematic configure information from the XML-based ECU configuration that is available on a generator engine. The generator engine analyzes ECU specific information and generates RTE modules.

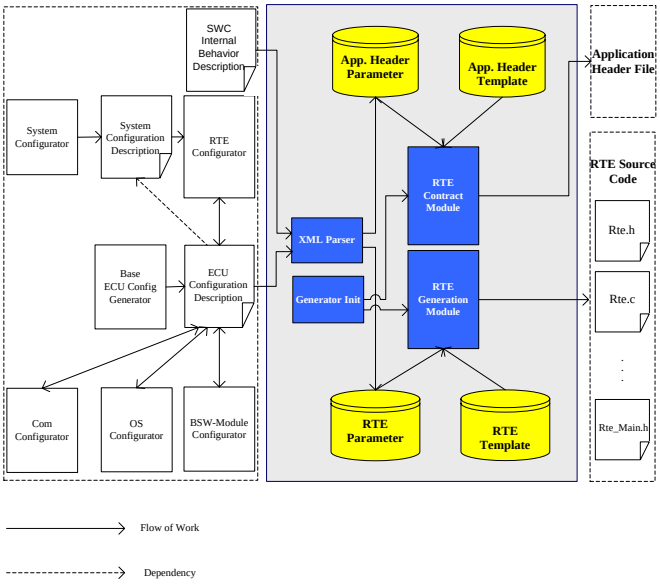


Figure 4. The Block Diagram of RTE Generator.

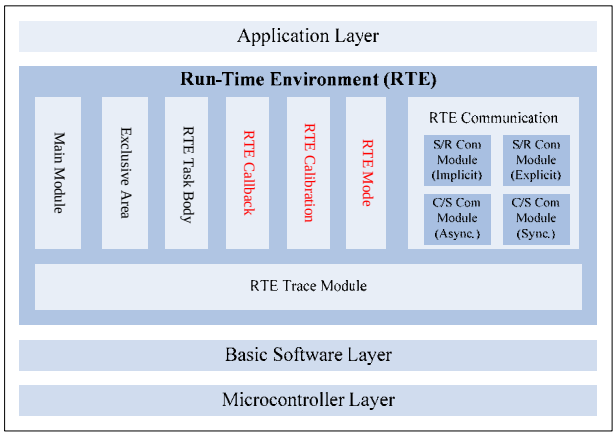


Figure 5. The Modules of RTE Template.

The RTE Template into the RTE Generator is modeled with the following RTE modules: (See Figure 5)

- RTE Communication
- RTE Trace
- RTE Main
- RTE Task Body/ Exclusive Area
- RTE Callback/ Calibration/ Mode

4 Test Development and Management

This section provides data for verification about the RTE Generator. The two step operations will be tested using a simplified model written to access all of the AUTOSAR functional requirements.

4.1 Some Definitions and Abbreviations

Here we present some definitions and abbreviations of the AUTOSAR standardization.

- AUTSOAR Software Component: The application software which has constraints imposed by the system designer
- Basic Software: Basic software module includes the OS, communication and several services. This module accesses the other basic software modules and ECU abstraction layer. AUTOSAR ECU contains software component as well as basic software module.
- Runnable Entity: Runnable entity means a sequence of instructions which is started by the run-time environment.
- Task Body: The term of task means a configuration object of the OS. The task body describes a piece of code.
- RTE Generator: The RTE Generator is the code generator tool which produces the RTE source code and API of the input information.
- RTE Template: The RTE Templates provide functionalities and API that is needed for the RTE modules. It enables generator engine to generate the RTE executable source code.

- API: Application Programming Interface
- BSW: Basic SoftWare
- SWC: SoftWare Component
- RTE: Run Time Environment
- VFB: Virtual Functional Bus
- XML: eXtensible Markup Language

4.2 Test Applications

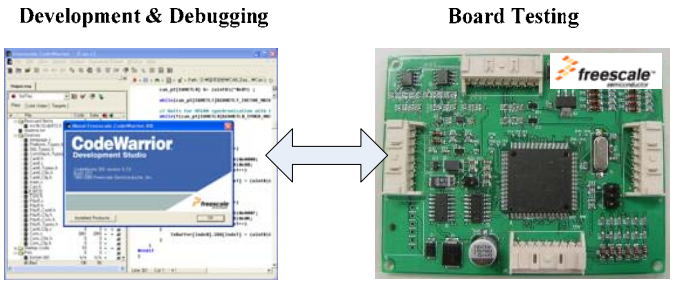


Figure 6. Simulation Modes: a two step process to verify generated code.

To verify the proposed approach, we will perform some experiments on the AUTOSAR platform. We shall design the application architecture of each ECU and generate the ECU specific code.

We used an example program. The example of application conducts whether the signal, as it should, transmits through the CAN (Controller Area Network) bus.

It is set as follows: The application architecture consists of two ECUs, which are connected to CAN-based transport protocol [11], [12]. One ECU is Node 1, and the other ECU is Node 2. (As depicted in the example software architecture, Figure 6) The Node 1 ECU contains Sender, Display 1 software components, and the Node 2 ECU has software components called Receiver and Display 2. The send data is triggered by the timing event. Then the signal is transmitted to the Display (ECU 1) and Receiver (ECU 2) software component. The internal communication between software components uses RTE channel.

The hardware configuration is implemented on the basis of the software structure as illustrated in Figure 6. The board used the S12X for the Freescale.

4.3 Integration Testing

TABLE I. TEST RESULTS

Functional Requriment Types	# of the Requriment Specification	Short Description	Test (P/F)
General Requirement	RTE00021	Per-ECU RTE customization	P
	RTE00065	Deterministic generation	P
	RTE00048	RTE Generator input	P
API	RTE00107	Information_type attribute	P
	RTE00108	Init_value attribute	P
	RTE00125	Interaction of 1:n communication	P

Test cases or test application example are used to exercise the instructions of the RTE function to detect standardization fault. Table 1 show the testing and verification tables using the generated RTE source code. Functional requirement are grouped under the two types, general requirement, API. The test results also include the short description over the AUTOSAR specifications requirements. RTE 00021, RTE 00065, RTE 00048 deals with general requirements types. The API function cover RTE 00107, RTE 00108, RTE 00125 requirements types.

5 Conclusions

In this paper, we verified a new approach to generate software components for ECUs automatically for the AUTOSAR based platform. Despite the rise of the development, few have attempted to address the AUTOSAR specifications. The major purpose of this study is to investigate development of the code generator that is based on the AUTOSAR standard.

We will extend the testing of the RTE generator module function in the future works. There remains additional research to be improved in accordance with the revision of the AUTOSAR RTE requirements.

6 Acknowledgment

This work was supported by the Basic Research Program of the Ministry of Education, Science and Technology.

7 References

[1] Leslie Lamport. "LaTeX: A Document Preparation System". Addison-Wesley Publishing Company, 1986.

[2] Ree Source Person. "Title of Research Paper"; name of journal (name of publisher of the journal), Vol. No., Issue No., Page numbers (eg.728—736), Month, and Year of publication (eg. Oct 2006).

[3] AUTOSAR, http://www.autosar.org

[4] AUTOSAR, AUTOSAR BSW & RTE Conformance Test Specification, 2009.

[5] AUTOSAR, Conformance Test Agency Accreditation, 2009.

[6] *CANoe 6.0*, Vector.

[7] A.L. Sangiovanni-Vincentelli, Di Natale. M, "Embedded System Design for Automotive Applications," Computer, IEEE, Vol. 40, Issue 10, pp. 42-51, Oct, 2007.

[8] G. Leen and D. Heffernan, "Expanding Automotive Electronic Systems," Computer, IEEE, pp. 88-93, Jan. 2002.

[9] Honekamp, U., "The Autosar XML Schema and Its Relevance for Autosar Tools," Software, IEEE, Vol. 26, Issue 4, pp. 73-76, 2009.

[10] B. Kienhuis et al., "A Methodology to Design Programmable Embedded System: The Y-Chart Approach," *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation*-SAMOS, LNCS 2268, E.F. Deprettere, J. Teich, and S. Vassiliadis, eds., Springer, pp. 18-37. 2002.

[11] N. Navet, Y. Q. Song, F. Simonot-Lion, and C. Wilwert, "Trend in automotive communication systems," *IEEE Special Issue Ind. Commun. Syst.*, vol. 93, pp. 1024-1223, June 2005.

[12] J. R. Pimentel, "An Incremental Approach to Task and Message Scheduling for AUTOSAR Based Distributed Automotive Applications," " in *Proc. of the 4th International Workshops on Software Engineering for Automotive Systems*, .IEEE, May, 2007.

[13] H. -C. Jo, S. –Q. Piao et al, "Automatic Source code generator based on AUTOSAR RTE Template for Vehicular Applications, " " in *Proc. of the ASME/IEEE International conference on Mechatronic and Embedded Systems and Applications*, San Diego, CA, Sep. 2009.

[14] Schreiner, D., Schordan, M., Goschka, K. M., "Component Based Middleware-Synthesis for AUTOSAR Basic Software, " " in *Proc. of the IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, pp. 236-243, Tokyo, Japan, March, 2009.

[15] Heinecke. H, Damm. W, Josko. B, Metz`ner. A, Kopetz. H, A.L. Sangiovanni-Vincentelli, Di Natale. M, "Software Components for Reliable Automotive Systems," *Design, Automation and Test in Europe 2008*, pp. 549-554, Mar. 2008.

[16] Schreiner, D., Goschka, K. M., "A Component Model for the AUTOSAR Virtual Function Bus," in *Proc. of the 31st Annual International Computer Software and Applications Conference*, Beijing, China, July, 2007.

[17] W. –B. See, P. –A. Hsiung, T. –Y. Lee and S. –J. Chen, "Software Platform for Embedded Software Development," in *Proc. of the Ninth International Conference on Real-Time and Embedded Computing Systems and Applications RTCSA-2003*, LNCS Vol.2968, pp. 545-557, Springer-Verlag, 2003.

[18] Dietmar Schreiner, Karl M. Goschka, "A Component Model for the AUTOSAR Virtual Function Bus," in *Proc. of the 31th Annual International Conference on Computer Software and Applications*, IEEE, pp. 635-641, July, 2007.

[19] Klaus Grimm, "Software Technology in an Automotive Company – Major Challenges," in *Proc. of the 25th International Conference on Software Engineering*, IEEE, pp. 498-503, May, 2003.

[20] H. -C. Jo, S. –Q. Piao et al, "Design of a Vehicular Code Generator for Distributed Automotive Systems, " in *Proc. of the Seventh International conference on Information Technology*, Las Vegas, NV, April, 2010

Requirements Engineering and Management for Software Product Line

Waraporn Jirapanthong
Faculty of Information Technology
Dhurakij Pundit University
Bangkok, Thailand
waraporn.jir@dpu.ac.th

Abstract*—*Software product line has been recognised as an important paradigm for software systems engineering. In the last years, a large number of methodologies and approaches have been proposed to support the development of software systems based on product line development. However, its context leads difficulties to software product line engineering in practical. It has been quested whether software product line-based approach is more productive and flexible than traditional software development model. This paper thus describes the experiences and challenges of requirements engineering and management that arise in the context of industrial software product line development. They have been derived from the study on empirical projects based on software product line and single software development.

Keywords: Software Product Line, Product Family, Requirements Engineering and Management, and Requirements Development

1 Introduction

Requirements management is concerned with understanding the goals of the organisation and its customers and the transformation of these goals into potential functions and constraints applicable to the development and evolution of products and services. It involves understanding the relationship between goals, functions and constraints in terms of the specification of products, including systems behaviour, and service definition.

The goals represent why a certain extent relates and what are in development terms. The specification provides the basis for analysing requirements, validating what stakeholders want, defining what needs to be delivered, and verifying the resultant developed product or service.

Requirements management aims to establish a common understanding between the customers and stakeholders and the project team that will be addressing the requirements at an early stage in the project life cycle and maintain control by establishing suitable baselines for both development and management use.

In addition to, many software development projects focus on customer satisfaction, quick adaptation to changes, and flexibility. Therefore, software product line development has become popular because it responds well to frequent changes in user requirements. Software product line shares a common set of features and are developed based on the reuse of core assets have been recognised as an important paradigm for software systems engineering. Recently, a large number of software systems are being developed and deployed in this way in order to reduce cost, effort, and time during system development. Various methodologies and approaches have been proposed to support the development of software systems based on software product line development.

However software product line development is criticized as having difficulties. Some difficulties are concerned with the (a) necessity of having a basic understanding of the variability consequences during the different development phases of software products, (b) necessity of establishing relationships between product members and product line artefacts, and relationships between product members artefacts, (c) poor support for capturing, designing, and representing requirements at the level of product line and the level of specific product members, (d) poor support for handling complex relations among product members, and (e) poor support for maintaining information about the development process.

This paper thus describes the experiences and challenges of requirements engineering and management for software product line, in comparison with single software systems.

2 Background of Requirements Engineering and Management for Software Product Line

Software product line systems share a common set of features and are developed based on the reuse of core assets. Many approaches [1],[2],[4],[7],[8] have been proposed to support software product line development. Those approaches mainly focus on *domain engineering* which is concerned the identification and analysis of commonality and variability principles among applications in a domain in order to engineer reusable and adaptable components and, therefore, support

product line development. There are three steps for domain engineering: (a) *domain analysis* is the process of identifying, collecting, organizing and representing the relevant information in a domain, based upon the study of existing systems and their developing histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain [7]. Software artefacts that are produced during the activity of domain analysis are called *reference requirements*, which define the products and their requirements in a family. The reference requirements contain commonality and variability of the product family. The activities occur during the domain analysis are scoping, defining of commonality and variability, and planning for product members and features.

(b) *domain design* is the process of developing a design model from the products of domain analysis and the knowledge gained from the study of software requirements or design reuse and generic architectures. Software artefacts that are produced during the activity of domain design are called *software product line architecture*, which forms the backbone of integrating software systems and consists of a set of decisions and interfaces which connect software components together. Software product line architecture differs from an architecture of single systems that it must represent the common design for all product members and variable design for specific product members. The activities occur during the domain design are defining and evaluation of software product line architecture.

(c) *domain implementation* is the process of identifying reusable components based on the domain model and generic architecture [3]. Software artefacts that are produced during the activity of domain implementation are called *reusable software components*. The activity is focused on the creation of reusable software components e.g. source codes and linking libraries that are later assembled for product members At the end of the domain engineering process, an organization is ready for developing product members.

Additionally, *application engineering* is a systematic process for the creation of a product member from the core assets created during the domain engineering. Domain engineering assures that the activities of analysis, design and implementation of a product family are thoroughly performed for all product members, while application engineering assures the reuse of the core assets of the product family for the creation of product members. There are activities such as: (i) *requirements engineering,* which is a process that consists of requirements elicitation, analysis, specification, verification, and management; (ii) *design analysis,* which is a process that is concerned with how the system functionality is to be provided by the different components of the system*;* and (iii) *integration and testing,* which is a process of taking reusable components then putting them together to build a complete system, and of testing if the system is working appropriately.

However, although the support for identifying and analysing common and variable aspects among applications and the engineering of reusable and adaptable components are important for software product line development, they are not easy tasks. This is mainly due to the large number and heterogeneity of documents generated during the development of product line systems.

For the development of single software system, the process has five major activities: (a) requirement definition, (b) software and system design, (c) implementation, (d) integration and testing, and (e) operation and maintenance. The documents are created for each single software system.

3 Research Method

This research conducted an experiment involving three software development projects that have similar requirements and some different requirements. A team of developers was required to achieve the software development projects two times: (i) to follow the software product line process, and (ii) to follow the single software process.

3.1. Empirical Project Development based on Software Product Line Process

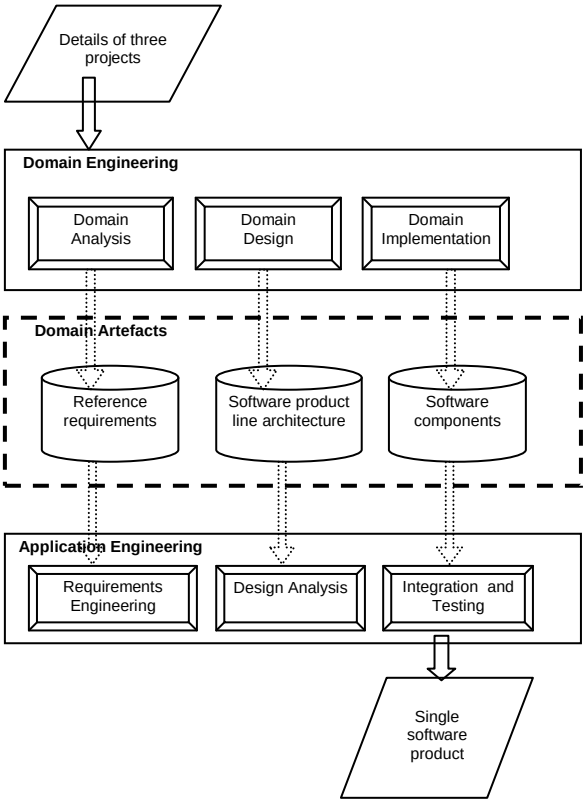


Figure 1. Software Produce Line Process

The project started with developers training in software product line processes and techniques. These developers were then tested for their understanding of software product line practices by using questionnaires. Those who passed the test were assumed to be ready to implement projects using software product line. At the beginning of a project the developers need to take several days to envision the high-level requirements and to understand the scope of the release. The goal of this activity is to find what the project is all about, not to document in detail. The developers then started developing a set of three projects by following the software product line practices. They

studied and analyzed all projects together and produced the software artefacts: (i) reference requirements; (ii) software product line architecture; and (iii) software components.

The artefacts were checked before submitting to the domain repository to be ready for application engineering process. Next, three software products were created based on the domain artefacts (i.e. reference requirements, software product line architecture, and software components). Before the software was accepted by customers, we ran test cases on the software. When the software passed all test cases, the projects are completed. The whole software product line process is shown in Figure 1. We then calculated and analyzed the qualitative and quantitative aspects of domain engineering process and application engineering process for each project. Then we checked the developers conform to software product line practices.

3.2. Empirical Project Development based on Single Software Process

For each project, developers divided their work based on their roles. Firstly, the developers summarized all requirements from customers and produced a user requirement specification. Next, they designed the system architecture, components and data models. They applied use case descriptions and diagrams to explain the requirements of each single software product. In addition, they also created class diagrams, sequence diagrams and activity diagrams of the entire project in this stage. They implemented the software by following the documents and used unit tests regularly. When completing all the components, the developers integrated all the pieces together and began an integration test. Finally, the developers delivered the customers the complete software when all of these stages finished. The artefacts that are checked and submitted to the repository are: (i) use case descriptions; (ii) use case diagrams; (iii) class diagrams; (iv) sequence diagrams; (v) activity diagrams; (vi) source code; (vii) testing documents; and (viii) coding standard and technical documents. All the end of this step, we calculated and analyzed the qualitative and quantitative aspects in each project.

4 Requirements Engineering and Management Development for Software Products

At the beginning of the experiment, the developers are given the description of artifact types which should be applied for requirement engineering process. However, in practical, there are several different types of requirements. Each modeling artifact has its strengths and weakness. Therefore several requirements modeling artefacts are applied. Table 1 summarises common artefacts for modeling requirements in projects.

Table 1. Common artefacts for modeling requirements

Artifact	**Type**	**Simple tool**	**Description**
Acceptance test	Either	Paper	Describes an observable feature of a system which is of interest to one or more project stakeholders.
Business rule definition	Behavioral	Index card	A business rule is an operating principle or policy that software must satisfy
Constraint definition	Either	Index card	A constraint is a restriction on the degree of freedom that a developer team have in providing a solution. Constraints are effectively global requirements for a project.
Data flow diagram (DFD)	Behavioral	Paper	A data-flow diagram (DFD) shows the movement of data within a system between processes, entities, and data stores. When modeling requirements a DFD can be used to model the context of the system, indicating the major external entities that the system interacts with.
Essential UI prototype	Either	Draft paper	An essential user interface (UI) prototype is a low-fidelity model, or prototype, of the UI for the system. It represents the general ideas behind the UI but not the exact details.

Essential use case	Behavioral	Paper	A use case is a sequence of actions that provides a measurable value to an actor. An essential use case is a simplified, abstract, generalized use case that captures the intentions of a user in a technology and implementation independent manner.
Feature	Either	Index card	A feature is a small useful result in the perspective view of users. A feature is a tiny characteristic of the system. It is understandable, and do-able.
Technical requirements	Non-behavioral	Index card	A technical requirement pertains to a non-functional aspect of the system, such as a performance related issue, a reliable issue, or technical environment issue.
Usage scenario	Behavioral	Index card	A usage scenario describes a single path of logic through one or more use cases or user stories. A use case scenario could represent the basic course of action.
Use case diagram	Behavioral	Draft paper	The use case diagram depicts a collection of use cases, actors, their associations , and optionally a system boundary box. When modeling requirements a use case diagram can be used to model the context of the system, indicating the major external entities that the system interacts with.
User story	Either	Index card	A user story is a reminder to have a conversation with the project stakeholders. User stories capture high-level requirements, including behavioral requirements, business rules, constraints, and technical requirements.

4.1. Requirement Development for Software Product Line and Single Software Systems

The projects have been developed based on study, analysis, and discussions of business domain. The team of developers analysed and designed a family of software systems with three members. Each member has shared and specialized functionalities with the family. The product members are aimed to satisfy different targets of customers.

According to several types of requirements artefacts as shown in Table 1, the specification of requirements are done in different documents. Particularly, the reference requirements is produced and documented in term of a feature model as software product line architecture is produced and documented in terms of subsystem, feature, and process models [5]. The feature model is created and composed of common features representing mandatory features, alternative and optional, representing different features between product members. The subsystem models is created and provides facilities for performing basic tasks in the systems. But there exist various instances of the process and module models, as well as there exist many instances of use cases, class, statechart, and sequence diagrams. The process models are created and each is refined for a subsystem in the subsystem model. The module models are created and each is refined for a process in the process models. Moreover, the artefacts of each product member are created. For example, a use case is used to elaborate the satisfaction of the functionalities for each product member. For single software development, a number of artefacts for each single software product are created during software development process. The artefacts of each single software product are usecase diagram, use case descriptions, class diagrams, statechart diagrams, sequence diagrams, and source code. Moreover, there are several techniques for eliciting requirements, summarized in Table 2.

Table 2. Techniques for eliciting requirements

Technique	**Description**	**Strength(s)**	**Weakness(es)**
Active stakeholder participation	Extends on-site user to have stakeholders (users) actively involved with the modeling of their requirements .	- Highly collaborative technique - Domain expert can define the requirements - Information is provided to the team in a timely manner - Decisions are made in a timely manner	- Many stakeholders need to learn modeling skills - Stakeholders are not available full time
Face-to-face Interview	Meets key stakeholders to discuss their requirements .	- Collaborative technique - Developers can elicit a lot of information quickly from a single person - Stakeholders can provide private information that they would not publicly tell	- Interviews must be schedules in advance - Interviewing skills are difficult to learn
Reading	A wealth of written information available from which developers can discern potential requiremetns or just to understand stakeholders better.	- Opportunity to learn the fundamentals of the domain before interacting with stakeholders	- Restricted interaction technique - Practical usually differs from what is writtern down - There are limits how much developers can read, and comprehend the information

4.2. Change Management for Software Product Line and Single Software Systems

According to software product line-based systems, new requirements management can be facilitated by the identification and analysis of commonality and variability principles among software product line and product members. A number of relations between artefacts are detected in order to determine the association between the new requirements and existing software artefacts in product member and software product line. Different types of traceability relations are created to identify the role of those relations [6].

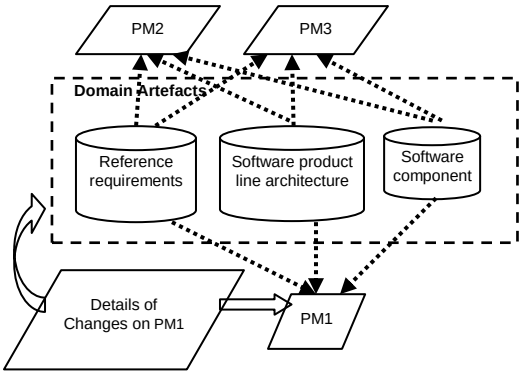


Figure 2. Empirical Project Maintenance

For the software product line-based systems, it is supposed the situation in which the organisation has established a software product line for their software systems with software product members. Those are created from the development phase. And the new requirements are done to a product member. Therefore, it is necessary to evaluate how these new requirements will affect the other artefacts of the product member and if these new requirements also affect other product members in the software product line that may be related to the new requirements. The artefacts are inspected and determined if they are related to the new requirements as shown in Figure 2.

For the single software systems, it is necessary to evaluate how these new requirements will affect any artefacts of each software product. Developers divided their work based on their roles. They reproduced new user requirement specification and redesigned the system architecture, components and data models. They applied use case descriptions and use case diagrams to explain the new requirements of the software product. They updated class diagrams, sequence diagrams and activity diagrams of the entire project in this stage. They re-implemented the software by following the documents and used unit tests regularly. When completing all the components, the developers integrated all the pieces together again and began an integration test. Finally, the developer delivered the customers the complete software when all of these stages finished.

5 Discussion and Conclusion

Requirements engineering and management is a central task of software product line development. It must be capable of deal with factors like upfront development of a domain model, the constant flow of requirements, a heterogeneous stakeholder community, a complex development organization, long-term release planning, demanding software architecture, and challenging testing processes. For successful software product line development, a collection of essential requirement development practices must be in place, which needs to support the meta project management capabilities. Many requirements engineering and management practices must be tailored appropriately to the specific demands of software product lines. The software engineering literature has pointed out the software product line development is more complex and demanding than single product development. This complexity has also particularly impact on requirements engineering and management. Of course, general challenges of requirements engineering and management also reoccur in software product line.

This work experienced the requirements engineering and management that arise in the context of industrial software product line development. We conducted the survey and interview. The developers are observed for the satisfaction regarding the process of software product line. It is found that the developers are satisfied the process that emphasis the software more than the documentation. However, the process would be difficult to inexperience developers and some experience developers tend to resist some software product line practices. According to the survey, it is found that 33% of developers tend to resist software product line practices with the above reasons, whereas 70% of developers are positive to using software product line practices. Particularly, 82% of developers are satisfied when performed the maintenance phase with software product line. Some of software product line artefacts are used during the maintenance phase. And it is satisfied by the developers. However, application engineering process depends on developer' skill. Moreover, some developers are unsatisfied to frequently update the documentation.

Additionally, the developer teams found that types of requirements can be separated into two categories: behavioural and non-behavioural. A behavioural requirements describes how a user will interact with a system concerning user interface issues, how a user will use a system or how a system fufills a business function or business rules. These are often referred to as functional requirements. A non- behavioural requirements describes a technical feature of a system, features typically pertaining to availability, security, performance, interoperability, dependability, and reliability. Non-behavioural requirements are often referred to as "non-functional" requirements. It is very important to understand that the distinction between behavioural and non-behavioural requirements is fuzzy. A performance requirement which describes the expected speed of data access is clearly technical in nature but will also be reflected in the response time of the user interface which affects usability and potential usage. Access control issues, such as who is allowed to access particular information, is clearly a behavioural requirement although they are generally considered to be a security issue which falls into the non-behavioral category. The critical thing is to identify and understand a given requirement. We found that it becomes an issue if the requirements are managed and mis-categorised.

Moreover, the results show that the effort metric of software product line-based projects is less than single software projects. Software product line-based projects enhance the productivity by using existing software artefacts. The methodology supports software reuse at the largest level of granularity. However, developers spent time and effort to establish domain artefacts. Also, some defects are discovered during the integration process for a product member. It took some effort to fix them. On the other hand, in the single software team, customers are involved at the inception of project determined requirements and contractual agreement. Developers wrote all documents before coding. Then customers changed some requirements, maybe after they acquired finally product, developers needed to significantly redesign and edit their documents. This took a lot of effort to achieve the task.

However, software product line is unsuitable for all projects. It serves the reuse practice in an organization having a large number of products, which have similar requirements and some differences. Developers must consider the characteristics of the project to ensure software product line is appropriate. In the other hand, waterfall process is suitable to serve a software project which is small and has solid requirements. Also, the developers are responsible for estimating the effort required to implement the requirements which they will work on. Although the developers may not have the requisite estimating skills, it does not take long for them to get better at estimating when they know and get familiar with the software process methods.

6 References

[1] Atkinson, C., J. Bayer, and D. Muthig. 2000. Component-based product line development: The KobrA approach. Pages 289-310. the 1st Software Product Line Conference, SPLC. Kluwer, Denver, Colorado, USA.

[2] Bayer, J., O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. 1999. PuLSE: A methodology to develop software product lines. Pages 122-131. the Fifth ACM SIGSOFT Symposium on Software Reusability (SSR'99), Los Angeles, CA, USA.

[3] Clements, P., and L. Northrop. 2004. A Framework for Software Product Lines Practice. http://www.sei.cmu.edu/productlines/framework.html

[4] Griss, M. L., J. Favaro, and M. d. Alessandro. 1998. Integrating feature modeling with the RSEB. Pages 76-85 in P. Devanbu and J. Poulin, eds. the 5th International

Conference on Software Reuse. IEEE Computer Society Press.

[5] Jirapanthong, W. 2008. An Approach to Software Artefact Specification for Supporting Product Line Systems. the 2008 International Conference on Software Engineering Research and Practice (SERP'08), Las Vegas, Nevada, USA, 2008.

[6] Jirapanthong, W., and A. Zisman. 2009. XTraQue: traceability for product line systems. Software and System Modeling 8(1): 117-144 (2009).

[7] Kang, K., S. Cohen, J. Hess, W. Novak, and A. Peterson. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

[8] Weiss, D. 1995. Software Synthesis: The FAST Process. the International Conference on Computing in High Energy Physics (CHEP), Rio de Janeiro, Brazil.

Characters of a Successful Project Team: Anatomy of a Capstone Software Engineering Project

Dr. Xiaohong (Sophie) Wang
Daniel Wheeler
Andrew Boyd
Omar Ejaz

Department of Mathematics and Computer Science
Salisbury University
1101 Camden Avenue, Salisbury
MD 21801 USA

Keywords:

Software engineering education, capstone projects.

Abstract

Real world projects are imminent part of an effective capstone software engineering course. As capstone projects grow larger and more sophisticated, software engineering practices to control the complexity are needed. Involving students in real world projects is a great way to teach valuable software engineering skills. In this paper we present our experience with an on-line multi-player game development project in our software engineering course. As in real world software development, there can be many factors that can affect the outcomes of a project. Our discussion and analysis focuses on what unique factors of student grouping affect the successful outcome of a student project.

1. Introduction

Modern software engineering education always enacts the software development theories in the context of reasonably large team projects. Through those large scale projects, the students can explore different approaches to managing the software process and gain insight into the complex cause and effect relationships underlying the process and the topics discussed in lectures are better enhanced through capstone projects.

One obvious benefit of real world projects for students is that those projects help them improve their self learning skills. The programming languages, software development tools and technologies required to complete those capstone projects are not necessarily taught and discussed in regular curricula. In many situations, the students have to conduct self-learning at the beginning of the projects. The capstone experience also enhanced the time and project management skills, enabled students to realize the importance of team work and became aware of the work ethics and professionalism in this field.

In the past ten years, our students in software engineering course have participated in many large real world software projects. It is widely known that many real world software development projects fail due to numerous reasons and student projects cannot escape this reality either. In our previous work, we have identified the factors affect the outcomes of a project within the unique framework of the on-campus environment [1,2]. We found that client involvement, project complexity, technology adopted, and student grouping are the determining factors for student projects. In this paper, we further explore this problem by analyzing our experience with an online multi-player game development project in our two-semester long capstone software engineering course during fall 2010/spring 2011. The game development project started with a vague set of product requirements technology to be used was given by the client. The students were given the freedom to use their imagination, creativity and skills to create the product. We try to identify, when client involvement, project complexity, technology adopted are fixed, what characters of a student team determine the success of a project.

The paper is organized as following: Section 2 discusses the background of this study. In Section 3, the anatomy of an online multi-player game

development project is performed. Finally, the summary and discussion is given in Section 4.

2. Background

In real world scenario, the outcomes of a software development project can be affected by many factors such as requirement changes, poor project management and incomplete process and quality control. In our previous work [1,2], we have identified the following factors affecting the success of a student project within the unique framework of the on-campus environment:

Client involvement

Among the successful projects, all the clients have been very active and responsive to the student team requests during the project development. On the contrary, among those failed projects, majority of the clients were either too busy to commit any time for the students, too pessimistic about the students capability, or not aware of the importance of client involvement. Although it is not surprising that good communication between clients and project team is a key factor to succeed in any project, it is proven to be vital to the success of student projects.

Project complexity

Due to the time limit and schedule constrains for two-semester software engineering courses, the complexity of a project is another factor that affects the success of a project. Good quality software takes time to develop. Even with strong technical skills, due to the project complexity, some projects may still fail. However, from a pedagogical point of view, the students benefit from the experience of working on successful as well as failed projects.

Technology

Technology adopted for a project has a huge impact on the success of the project. The students often will have to learn the new technologies on their own when they work on a project and the self-learning ability determines how long the learning curve is. Strong learning and research capability is a crucial for student projects to success.

Student grouping

Building a project team is very important for real world projects. However, when building a student project team, besides the technical strength and background of each member and the team dynamics, the learning aspect is often taken into account in an education environment. Evenly distributing students with mixed skills among each team is a risk since the weaker students might drag down the project development or even cause project failure. From a pedagogical point of view, the benefits can far outweigh the risk. The students can learn from this experience on how to achieve the most when given limited resource.

In this study, we are interested in discovering what specific characters about a project team can affect the success of a student group project.

3. Project case study

In this section, we describe an on-line multi-player game development project during the two-semester software engineering course.

Games are excellent projects in that they incorporate a wide range of techniques from computer science curriculum. From designing efficient data structures, effective memory management, intuitive user interfaces and rendering displays, networking and concurrency, accessing large data sets and managing system resources, as well as autonomous game entities and artificial intelligence. All of these elements need to function correctly and work together in real-time. Using game development projects also can capitalize on students' interests. Students are motivated to work on games, and they are domain experts. They understand games and they are their own best critics. Game projects allow students to exercise their creativity in ways not typically challenged in the computer science curriculum. Finally the involvement and the excitement of the accomplishments provide unique benefits for using game development as capstone project. A positive stimulating experience is created when the computer games are shared with their friends, families, and prospective employers or even sell it to potential clients.

Problem statement and challenges

A couple of years ago, a local resort called Frontier Town expressed their interests in hosting an on-line multi-player game with Western theme. To encourage the exercise of the creativity of game developers, the client did not give much specification of the game. Instead, the client indicated that a game that is similar to the popular Club Penguin game would be ideal. Club Penguin is a popular online multi-player game that consists of many penguin avatars wandering around their game world, playing mini-games and finding items to customize their characters. The online multi-player game restricts the technical architecture of the Frontier Town application to be client-server based. To support cross platform capability, the client needs

to be developed in Java. Two-dimensional graphics is used in the application.

During fall 2010 and spring 2011, a student project team in software engineering course has been assigned to work on the Frontier Town game. When the team members began working on Frontier Town project, little did they realize how big of a jump it was going from a concept-based class to the harsh realities of software development. Previously all of the difficult work in programming had been done for them – they were given libraries, API's , documentation and full help from an instructor when needed. However, that handholding ended in the first week of the project when they were assigned the task of creating a social networking game. The team faced the following challenges as well – none of the team members had any extensive experience with Java programming, graphics programming, and any client-server application and multi-player online game development. Most importantly, none of the team members had artistic background, which is critical in creating a successful game to attract a large user base.

Next we describe how the team dealt with those challenges during the game development with respect to their research and learning approaches and the contribution of the team leader and team members.

Team Structure

The team consists of five students senior computer science students. Table I shows their team structure and the role and responsibilities of each person. DW volunteered to be the team leader at the beginning of the semester.

Team member	Role and responsibilities
DW	Team leader, Programmer *game backend, graphics*
OE	Programmer *minigames, website*
AB	Programmer *networking*
JC	Programmer *mini-game, testing plan*
GB	Programmer *graphics, testing plan*

Table I. Team Structure

The team's regular weekly meeting was scheduled on Wednesdays and team communication was achieved through informal subgroup meetings, e-mail message exchange, and a web-based team bulletin board system.

Research and learning approach

When facing the challenges of no experience with Java, graphics, online game development, the team started by conducting self-learning using web-based materials, books, and seeking advises and help from other fellow students with those knowledge and experiences. The team also adopted learning by doing approach and began to build some pieces of the game when learning to programming Java and 2D graphics. In fact they had been using this approach throughout the project development.

Team leadership

As a team leader, DW had to view and place his members wisely in order to ensure efficiency and quality. He had to gauge what each team member was capable of doing. This was a challenge primarily due to the fact that none of the members had any experience programming in the Java and game development; this was the primary predicament throughout the course of the project. In addition, internal dissatisfaction was also troubling to the team leader. Two of the group members would not be able to produce any significant amount of work in time with good quality and the rest of the team put one hundred and ten percent into the project.

Besides the project management tasks, DW primary task was to integrate and complete the main game backend and work hand in hand with other team members in order to ensure the integrity and consistency of the game. DW also completed the graphics for all three of the main game areas (Figure 1), water (Figure 2), the fisherman NPC, and loading screen, and horse game within a very small window of time. He developed the Horse Racing mini game and the whole main frame of the game which includes collision, key events, transitioning, and animation.

Figure 1. Main Scene

Figure 2. Water

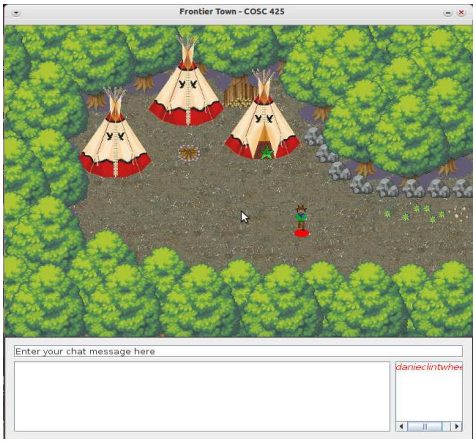


Figure 3. Indian Tipi

Team member contribution

The very first thing that the team did was to decide upon each member's role in the upcoming project.

During the project development, AB worked mainly on the networking portion of the game. At the start of the project, his main focus was adding a player list that would show concurrent online players in the same zone as the local player. The networking portion of this task was done fairly quickly and a graphical representation to show the player information in a meaningful manner was added. To make sure the player list was updated and offline players were not listed, AB had to add an additional network event that triggered a broadcast from the server to all the connected clients to notify them whenever a player disconnected. This made the player list dynamic. The disconnection event was also used to prevent offline characters from being painted. AB had managed to implement the majority of the features such as multiple players and chat in the first semester so his main focus was on improving networking performance for the second semester. After trying several methods and doing research into the performance problem as well as consulting with experienced fellow students, AB implemented finding a good value for packet throttling, thread cleanup and socket disconnection to gave the server better stability. AB's other contributions include the final graphics of the inside of the bank (Figure 4) and Shorty's place, the collision for the inside of both of these places. At the end he also added a random weather feature that made alternated between normal and rainy weather.

Figure 4. Inside Bank

OE's primary role was to add interactivity into the game through a series of minigames. He designed four minigames: saloon shootout, trivia challenge, blackjack and fishing. OE also created the website and java launcher for the game. During development, each minigame presented its own unique challenge. Saloon shootout (Figure 5) was the first game OE created. This was challenging in its own way because OE had little experience with Java and was essentially creating a game inside of a game. During the development, the major issue OE ran into was adapting a mouse listener to the game. Having a game that runs on mouse clicks requires an entirely different type of logic. This was eventually resolved with carefully planning the logic before actually programming it.

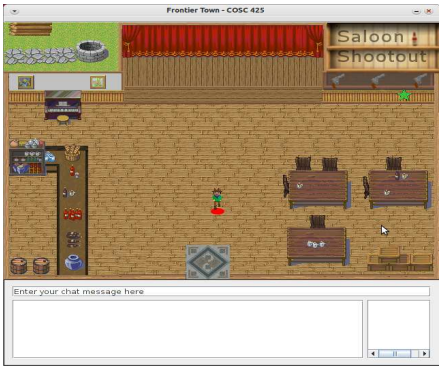


Figure 5. Saloon and shoot game

Because of the steep learning hurdle, it took OE the entire first semester to produce the finished product.

Blackjack (Figure 6) is by far the most involved game in terms of logic. It required very careful planning because of the complexity of the game. There were also some artificial limitations that had to be implemented. For example, in real blackjack, a player can theoretically have as many cards as needed unless certain conditions are met. This could not be allowed in the game because of limited screen size (and in order to maintain consistency). A workaround for this was to add a limit of 5 cards for both the dealer and player. This posed a challenge because it required an additional layer of complexity around all game stopping conditions – the game would also have to check the number of cards in addition to their value.

Figure 6. Beach scene

While Blackjack was the most complex game, fishing game (Figure 7) turned out to be the biggest technical challenge to overcome. The fishing game was simplest in nature – just have a hook catch a single feature and bring it back up to the boat. However, it was by far the most cumbersome underneath the hood. There are so many things going on in the game simultaneously that is not obvious at first. The primary technical challenge encountered was threading – not for performance reasons but to have multiple fish at the same time. Threading was something that OE had no experience and no background upon. It took days of experimenting and testing code before multiple fish were on the screen. In addition, the game had to be able to manage a dynamically changing amount of fish that would be randomly generated – in both location and frequency. Variable speed was also added to the fish for added difficulty. In order to overcome this, OE spoke with fellow students about the concept of threading and spent time to understand the concepts. He implemented threading in the music class in order to gain the skills necessary before doing the much more difficult fishing threading. Ultimately, his efforts were successful and the game was created with all intended features working.

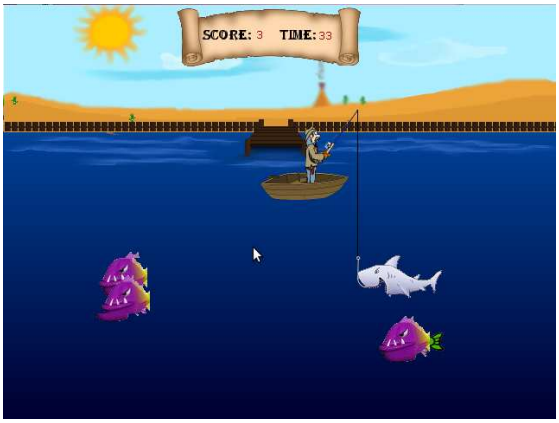


Figure 7. Fishing game

OE also created the Frontier Village website. It was important to create an attractive website that would draw visitors to it but also be functional at the same time. With this in mind he used JNLP, a built in launcher that automatically downloads and runs all required resources on a user's computer.

The other two team members, GB and JC involved a mini-game development, graphics design as well as the development of test plan and user manual. Unfortunately, as the semesters progressed, it was realized that GB and JC would not be able to produce any work with enough quality to make it into the final product. GB was picked as a team member for the second semester because his previous team got dissolved at the end of the first semester. JC simply did not put forward enough efforts in the project. Throughout the two semesters, he has not been able to contribute a single line of code or image to the final project.

4. Summary and Discussion

Over the course of this project the team learned several important lessons about all aspects of software development: both technical and non-technical.

Reflecting on the two semesters, the group is satisfied with our final project. Despite the unfortunate setbacks with two of the team members, they were able to work well around those problems. Their final product met the primary requirements of creating a social networking game with the intended western theme. All the major original features they planned to have at the beginning of the project such as chat, player list, multiplayer were completed to their satisfaction. In addition, some extra features such as minigames, updated graphics, NPC were also implemented in the game. An added bonus was that at the end of the semester, their project reached a state that they presented and attempted to sell to the client.

The team members' efforts and eagerness to work hand in hand through a challenging year of design, research, planning, implementation and analysis is the key to the success of this project. The group grabbed the task by the horns and placed every ounce of heart and effort into its completion and success. In addition, this experience has provided each team member with a technical outlook in regard to object oriented programming, integration, inheritance, polymorphism, GUI, JAVA, event driven programming and software design. From the independent decision making aspect, they learnt a lot in conducting independent research, comparing and testing different methodologies and making the best choice under each scenario.

From the project management aspect, a strong team leader, who was motivated, organized and responsible, was the crucial factor for the success. Knowing that a strong team was essential to success, the team leader made sure to keep close tabs on the group. He provided the team with a strong leader and eagerness to fix and solve problems. When conditions were difficult, he also gave a proverbial splash of colder water in order to encourage the team. From the team leader point of view, leading a project of this size has provided him with a stronger degree of patience, a willingness to truly understand an individual, and an eagerness to work with others in order to complete a defined task. In addition, the team leader is truly able to distinguish the character, ethics and traits that make up an individual when working with someone on a year-long project.

5. Acknowledgements

This project is also partially supported by Welcome Fellowship provided by the State of Maryland Higher Education Commission, USA.

6. References

[1] Lu, H. and X. Wang, 2007: "Game Development Projects in Software Engineering Course", *Proceedings of the International Conference on Software Engineering Research and Practice*, Las Vegas, Nevada, June 2007, 511-518.

[2] Wang, X., D. Spickler and B. Wainwright, 2010: "On-campus Collaborative Project experience for Software Engineering Course", *Proceedings of the International Conference on Software Engineering Research and Practice,* Las Vegas, Nevada, July 2010, 446-451.

On using high-level structured queries for integrating deep-web information sources

Carlos R. Rivero
University of Sevilla, Spain
carlosrivero@us.es

Rafael Z. Frantz
Uniju, Brasil
rzfrantz@unijui.edu.br

David Ruiz
University of Sevilla, Spain
druiz@us.es

Rafael Corchuelo
University of Sevilla, Spain
corchu@us.es

***Abstract*—The actual value of the Deep Web comes from integrating the data its applications provide. Such applications offer human-oriented search forms as their entry points, and there exists a number of tools that are used to fill them in and retrieve the resulting pages programmatically. Solution that rely on these tools are usually costly, which motivated a number of researchers to work on virtual integration, also known as metasearch. Virtual integration abstracts away from actual search forms by providing a unified search form, i.e., a programmer fills it in and the virtual integration system translates it into the application search forms. We argue that virtual integration costs might be reduced further if another abstraction level is provided by issuing structured queries in high-level languages such as SQL, XQuery or SPARQL; this helps abstract away from search forms. As far as we know, there is not a proposal in the literature that addresses this problem. In this paper, we propose a reference framework called IntegraWeb to solve the problems of using high-level structured queries to perform deep-web data integration. Furthermore, we provide a comprehensive report on existing proposals from the database integration and the Deep Web research fields, which can be used in combination to address our problem within the previous reference framework.**

***Index Terms*—Internet and emerging technologies; Semantic Web.**

I. Introduction

The Deep Web is composed of millions of applications that provide valuable data, which is usually served by querying search forms coded in HTML [4], [15], [52]. There are a number of studies in the bibliography about the Deep Web, which state a growth in the number of deep-web applications. Bergman's report [4] estimated 200 000 applications in 2001, Chang et al. [15] estimated 307 000 applications in 2004 and, finally, Madhavan et al. [52] estimated 25 million of deep-web applications in 2007.

Our research focus on the usage of high-level structured queries to integrate the deep-web data, which may help reduce the cost of a deep-web data integration solution.

To integrate the deep-web data is crucial but challenging since its data is behind search forms [50], which are designed by and for users and they do not have formally-defined semantics. Virtual integration (also known as metasearch [16], [37]) is a technique to perform deep-web data integration [52].

Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

Virtual integration approaches provide a unified search form for a specific domain, e.g., travels, hotels or flights. This search form is built by merging search forms of the same or related domains [35], [37]; the user fills it in, and the system translates the unified search form to fill the application search forms in.

Virtual integration approaches issue queries through search forms by means of the field values of the unified search form, which has proved to reduce integration costs [16], [37]. Increasing the abstraction level using high-level structured query languages may indeed help reduce integration costs. In the virtual integration approaches, the unified search form abstracts away from the actual applications, dealing with the query capabilities of the application search forms. High-level structured queries abstract away even from the unified search form, which has also its own query capabilities and do not have formally-defined semantics: the developer of a integration solution is only concerned with developing appropriate queries, which are posed over the solution. Note that the unified search form allows more specific queries than keyword-based query interfaces, and high-level structured languages allow even more specific and complex queries.

In this paper, we propose to use high-level structured queries to perform deep-web data integration, and report on a reference framework called IntegraWeb that combines results from the database integration and the Deep Web research fields. Specifically, database integration techniques provide the architecture of the solution. Deep-web virtual integration approaches are used to deal with the search forms of the deep-web applications. Finally, both deep-web surfacing and virtual integration approaches retrieve data pages and, in combination with information extraction and ontologising techniques, extract structured data from these pages.

This paper is organised as follows: Section II presents the research efforts made on the database integration and the Deep Web research fields. In Section III, we present IntegraWeb and we survey the state of the art in deep-web data integration. We conclude with Section IV, which presents our conclusions.

II. Research on data integration

In the last 10-15 years, data integration has been a very active research field [33]. The first approaches to data integration were related to database and, in next years, these approaches have been converging to the Web. Note that high-level structured queries can be specified in a number of

languages, e.g., SQL or XQuery [7] are used in the context of database, and SPARQL [61] in the Semantic Web arena.

Database integration techniques focus on providing unified access to a number of applications. Each application has one or more models of its data, which is commonly called schemes. Schemes are comprised of schema attributes, which are the properties of the schema, e.g., the departure date of a flight. Amongst the schemes there are a number of semantic mappings ("semantic glue"), which are formulae that provide the semantic relationships between the schema attributes [51], [54].

The existing approaches in the database integration research field are Mediators [26], [33], Peer Data Management Systems (PDMS) [20] and PayGo systems [52], [68].

Mediators offer a mediated schema to the schemes of the integrated applications (known as application schemes), and they need wrappers to provide access to them. Mediators deal with the translation of the user query by means of the semantic mappings. The user query is posed over the mediated schema and it is translated into some queries over the suitable application schemes. Finally, the data of the different applications is combined to give an answer to the user. TSIMMIS [26] is an example of a Mediator.

Peer Data Management Systems (or PDMS) are the next step in database integration systems after Mediators. The mediated schema in Mediators is actually a bottleneck in the process [31], [32]. Instead of having a unique mediated schema, the applications in PDMS have their own schema and there are mappings between these schemes. An example of a PDMS is Piazza [30].

PayGo systems are the next step after PDMS and they have a gradually increasing presence in the database integration community [22], [50], [52], [68]. PayGo systems are inspired in the concept of Dataspaces [22], [29] that do not require full semantic integration of the applications to provide integration services. These systems offer one schema for each application (as in PDMS) or a mediated schema (as in Mediators) but there is a key difference: the uncertainty. Instead of full semantic mappings, PayGo systems use probabilistic mappings and schemes. An example of PayGo systems is UDI [68].

The existing techniques that are used to have access to the Deep Web are surfacing [2], [43], [53], [65], [71] and virtual integration [16], [37].

Deep-web surfacing approaches (also known as crawling [2], [43], [53], [65], [71]) provide automatic access to the Deep Web. These approaches expose deep-web pages behind search forms and index them in search engines. These techniques precompute the most relevant form submissions for the selected search forms. These systems do not have to cope with the problem of building schemes, instead of this, the challenge is to automatically generate relevant form submissions that retrieve significant data pages. Google's Deep Web Crawl is an example of a deep-web surfacing system [53].

Virtual integration approaches work similarly to Mediators. They offer a unified search form that is created by using the search forms of the integrated applications, which are usually of the same or related domains. The user fills the unified search form in and it is used to fill the search forms of the different deep-web applications in. After submitting a filled form, a result page is obtained, and it is typically a list of links to data pages, e.g., a list of books in Amazon. Finally, data pages have to be retrieved. An example of a deep-web virtual integration system is MetaQuerier [16].

III. IntegraWeb

IntegraWeb is our contribution to combine the database integration and the Deep Web techniques. IntegraWeb is a reference framework that allows to integrate deep-web data using a high-level structured query language, such as SQL, XQuery or SPARQL.

In Figure 1, we present an example of deep-web data integration that consists of a unique entry point, which models information about travels (virtual schema), and it integrates two application schemes that models information about flights and hotels (application schemes). The Query Processing task takes the user query over the virtual schema as input and translates it into a number of queries to the application schemes, in the example, the user wishes to start the travel on October, 5 and the price must not exceed 3500€. This query is translated into a number of queries for flights and hotels.

The Search Form Processing task takes one application query as input and it calculates a number of search form submissions that has to performed to answer the application query. In our example, for flights, an origin and a destination airport need to be filled in; note that if the application query specifies the name of the city, it has to be transformed into the airport(s) of this city. The Deep Web Accessing task deals with the extraction of deep-web application instances by filling the search forms in, navigating through the result pages and extracting and giving semantics to the data pages. Finally, the instances have to be filtered and compiled in both Search Form Processing and Query Processing tasks.

In the next sections, we describe the processes performed by each task in the IntregraWeb reference framework, and we survey the most related literature.

A. Query processing

The Query Planner (cf. Figure 2) takes a high-level structured user query as input, this query has to be expressed in terms of a virtual schema, and it generates an execution plan by using the mappings involving other schemes. The execution plan consists of a number of queries, each of which is related to one application schema only. The execution of the queries can be ordered (i.e., the results of a query are used by another query) or parallel (i.e., independent queries). The Plan Executor takes an execution plan as input and iterates until it is finished. In each iteration, a query over an application schema is processed. Finally, the Query Compiler combines the intermediate instances retrieved from the applications using the execution plan and returns the results to the user. These intermediate instances may need filtering.

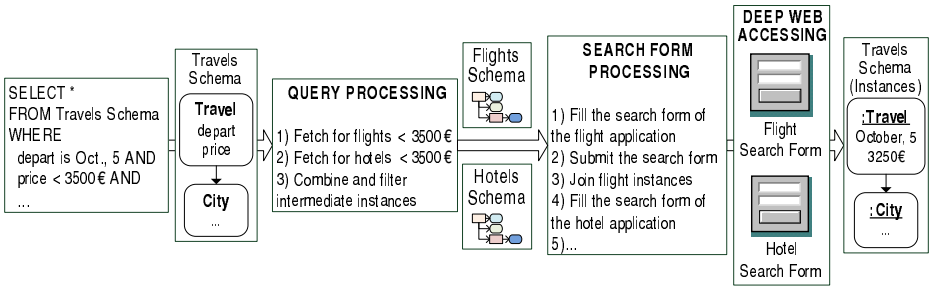


Fig. 1. An example of deep-web data integration

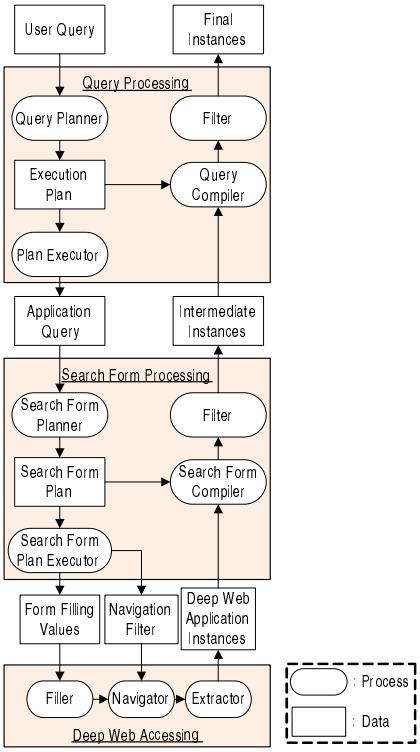


Fig. 2. The IntegraWeb reference framework

In the Query Processing task of our example (cf. Figure 1), the execution plan could be parallel, if the applications do not have any dependency between them, or could be ordered and it retrieves firstly flight instances and then hotel instances, e.g., the departure flight departs at October, 5 and arrives at the destination city at October, 6, the idea is to book the hotel the same day we arrive at the destination city. Therefore, in this example, the arrival date at the hotel needs to be October, 6 but note that we start the journey on October, 5. There exists a dependency between the flight application and the hotel application. To the best of our knowledge, there is not a proposal that takes this problem into account, i.e., the dependency between two (or more) application schemes.

In this module, any database integration technique is suitable. When using Mediators, the system is configured having a mediated schema and a number of application schemes. In Mediators, mappings are defined mainly using two techniques: Global-as-View (GaV) and Local-as-View (LaV) [33]. In GaV, the mediated schema is defined in terms of views over application schemes and, in LaV, application schemes are defined in terms of views over the mediated schema [28], [45]. Another technique is GLaV [24] that combines the advantages of LaV and GaV. A query posed over a GaV system is answered by query unfolding [9] and, in LaV systems, by query answering [28]. Another technique to answer a query over a mediated schema is to approximate it to the application schemes [14].

In other systems, such as PDMS or PayGo, the semantic mappings are specified by using GaV, LaV, GLaV. Generating those mappings is a labour-intensive and error prone task, so it is needed tools that help user to build and maintain them or tools that generate them automatically. The Clio system is an example of a tool that helps users to specify their mappings [27].

The Clio system requires a user to devise the mappings. The process of inferring such mappings automatically is commonly referred to as schema matching. Rahm and Bernstein [66] surveyed the schema matching techniques in 2001, a more recent survey is presented in [21]. Schema matching is a very active research field, but it is still an open problem because of an unavoidable consequence of ambiguity in the meaning of the data to be integrated [5]. Therefore, the intervention of a human is needed.

According to Bernstein and Melnik [5], the solution is to raise the level of abstraction in which mappings are specified, and they propose the model management technique that supports working with mappings between schemes in a high abstraction level. Another challenge in the semantic mappings field is to work with uncertainty, an approach is probabilistic semantic mappings and PayGo systems support them [68].

Finally, the Semantic Web research field is facing up the problem of querying distributed applications, and there are some approaches that translate a user query into a number of queries that are issued to multiple applications. Quilitz and Leser [63] or Langegger et al. [44] divide a SPARQL user query into a number of queries that are issued to multiple SPARQL endpoints. User queries are issued to a global schema that is comprised of multiple endpoints, which are defined as views over the global schema. Note that in the Semantic Web context, the recommendation of the W3C for the high-level

structured query language is SPARQL [61].

B. Search Form processing

The Search Form Planner (cf. Figure 2) takes an application query as input, which is expressed in terms of one application schema, and generates a search form plan that consists of a number of search form submissions, i.e, a number of form filling values which are taken from the query that is being analysed, and a filter for each submission, this filter is applied during navigation.

The Search Form Plan Executor takes a search form plan and feeds the Filler with form filling values and the Navigator with the appropriate filter. Note that query values could need some transformations to perform form filling. There can exist schema attributes that do not correspond to any fields in the search form. The constraints in a query that refer to those attributes must be grouped and applied as a post-filter to the results the search form returns. This process is performed by the Search Form Compiler, which is also responsible for combining the result instances retrieved by the Extractor, using the search form plan.

Existing approaches model a search form as a parameterised view over the application schema [59]. In this context, to have an answer for an application query, we can use the techniques for answering queries using views [28] (cf. Section III-A). A key concept in this module is query feasibility: it analyses whether a query can be issued over a schema without executing it. This analysis avoids a trial and error process in which the system poses a query and executes it until reach a suitable query. Petropoulos et al. [60] presented CLIDE, a user interface for building SQL queries over a set of parameterised views. CLIDE, besides query building, warns users when a query over the selected views is not feasible.

Pan et al. report on [59] a generic framework for representing query capabilities. This framework analyses the feasibility of SQL queries over applications that include deep-web applications. An implementation of this framework and a recap on its main drawbacks is presented in [67].

In deep-web virtual integration, some approaches deal with the possibility of filling more than one search form in, having only one query. For example, if we have a search form of flights in which we have to select a price range between 0-1500€, 1500-5000€ and >5000€, and a user query is of the form "price < 3500€", we need to fill two search forms in, one with 0-1500€ and another with 1500-5000€. In this last case, a filter of flights whose price is less than 3500€ is needed. Zhang et al. [76] developed a translation technique, from the unified search form to application search forms, that allows multiple form submissions.

C. Deep Web accessing

The Filler (cf. Figure 2) takes a number of form filling values as input and fills the application search form in. After filling, this search form is submitted and the resulting page is sent to the Navigator. The Navigator takes the result page and classifies it into three categories: error, data or list. An error page finishes the process, a data page is returned immediately and, if the result page is a list page, the Navigator navigates to the data pages by clicking on suitable links. Some optimisations can be done in this process, for example, if we are looking for flights whose cost is less than 3500€, we can avoid clicking on flights whose price is higher. To perform this optimisation, the Navigator uses the filters given as input by the Search Form Plan Executor.

Finally, the resulting data pages are returned to the Extractor. The Extractor takes these data pages as input and produces intermediate instances. This task is performed by extracting information of the web page and giving semantics to this information.

Regarding form filling, it is needed a search form model to give semantics to search forms, which are designed by and for users. Deep-web approaches use different types of search form models [2], [36], [43], [55], [65], [75]. The first step to generate a search form model is to identify labels, i.e., text strings that give users an intuition about the semantics of a form field [2], [36], [39], [43], [55], [58], [65], [75].

There are three different approaches to identify form field labels automatically, and they rely on the idea that the label positions in a search form have significant semantic information. In textual identification [36], [39], [43], the HTML code of a search form is used to extract field labels. These techniques rely on the idea that analysing HTML code approximately captures the visual layout. In layout position techniques [2], [55], [65], [75], besides the HTML code, physical layout is used to extract field labels. In machine learning approaches, a variety of algorithms are combined to identify field labels [58].

The next step is to identify hidden database attributes. Search forms issue queries to deep-web databases whose structure is hidden partially, since a quick glance at the results of submitting a search form can reveal some of their attributes. These techniques extract hidden database attributes using only a search form. Some approaches, after label extraction, identify fields as attributes [2], [43], [65]. Other approaches allow attributes comprised of form fields with their associated labels [36], [55], [75], [76]. Kushmerick uses a machine learning algorithm to perform attribute extraction [41]. Some form models offer more information than labels and attributes, e.g., field order in form filling [25], mandatory fields [69], attribute logic relationships or attribute units [36], and search form query capabilities [69], [75].

Search form query capabilities are the different modes of querying a search form, e.g., a search form of books accepts queries by title, which is mandatory, author, publication year or any combination of them. The proposals that rely on an advanced search form model extract the search form query capabilities [69], [75], [76]. Shu et al. [69] extracts them by issuing predefined queries through search forms that help detect mandatory fields. Zhang et al. [75], [76] extract hidden database attributes, operators that are applied to these attributes, and their ranges. Attributes are combined by conjunctive queries because this is enough to capture the

query capabilities of most deep-web applications.

Regarding navigation, an important task is to classify the page that results from a search form submission. This result page can be an error page, a data page or a list page; in the last case, it is necessary to navigate through the page links to access to the data pages, which have to be also classified. Classifiers of web pages use a number of features that can be from the textual content, structural content or the visual layout [62], [71]. Note that the classifiers usually work with the significant portion of a web page that is the piece that results from removing miscellaneous headers and footers.

Textual content techniques study features such as predefined text patterns [47], [64], [69], e.g., "No matches" for error pages or "Showing 1 - 20 of 50 000" for list pages. Structural content approaches use the HTML tag tree to extract the features [6], [10], [71], e.g., the position of a label in the tag tree have significant meaning, if the label is present in the significant portion of the web page it has to be considered. Visual layout techniques use the features from the visual representation rendered by a web browser [2], [39], [64], [75], e.g., a web page with a number of images and some text at the right side of each image can be a list page that shows title pages of books and their corresponding information.

Caverlee and Liu [10] combines textual and structural techniques to perform web page classification. This approach fetches for areas in web pages that can be used to answer a user query. At the beginning, they cluster web pages according to their structural layout and, after this process, they filter each cluster using textual features such as the size of the web page.

A web page can contain some features that are missing, misleading, or unrecognisable for various reasons, e.g., the web page contains a number of large images or Flash objects but too little textual content. In these cases, the neighbours of the web page are used to supply supplementary information for the classification process [62]. A neighbour of a web page A is another web page B that has a link of the form $A \rightarrow B$ or $A \leftarrow B$ whose distance is one. Distances greater than one can be used to perform the neighbour analysis [62]. Besides the links between web pages, there are a number of approaches that use artificial links such as textual similarities between the pages or the pages that co-occur in top query results [62].

Navigation patterns are used to navigate through links [43], [56], [57], they define common navigation paths that are repeated in several deep-web applications. Another technique that can also be applied to navigate through links is focused crawling [11]. Focused crawling techniques or those proposals that use navigation patterns to optimise navigation processes rely on a blind search, which results in unnecessary clicks that lead to uninteresting pages. This argues for a more intelligent method to navigate through deep-web applications.

One challenge is to avoid these excessive clicks by identifying summary information of interest in list pages, and extracting this information so that uninteresting links are not clicked. Summary information has to be detected using record extraction techniques [1], [40], [46], [74], [77], and data extraction can be performed by information extractors. Montoto et al. [56], [57] presented a workflow language that allows to define a task to not perform a blind search.

Regarding extracting, there are four types of information extraction: hand-crafted, supervised, little supervised and unsupervised. Hand-crafted extractors rely on the user to provide the extraction rules [17], [34], [48]. Supervised extractors use a set of labeled documents to learn extraction rules using induction procedures [8], [23], [42]. Little supervised extractors work the same as supervised ones but with just one labeled document [13], [38]. Unsupervised extractors learn the extraction rules without requiring a set of labeled documents [18], [49], [72], [74]. Turmo et al. [70] survey information extractors that work on natural language data pages and Chia-Hui Chang et al. [12] survey information extractors that work on structured and semi-structured data pages. Information extraction techniques extract pieces of text from data pages; later they must be endowed with semantics by means of ontologisers [3], [19], [73].

IV. Conclusions

Deep-web virtual integration approaches reduce the integration costs by providing a unified search form, which abstracts away from the actual search forms. We argue that increasing the abstraction level may help reduce the cost of a deep-web data integration solution even further. This new abstraction consists of working with high-level structured queries that are posed over the integration solution: high-level structured queries allow to abstract away from the deep-web application search forms, and even from the unified search forms. Also, high-level structured languages allows more specific and complex queries than the unified search forms or keyword-based query interfaces.

We believe that this new abstraction level avoids the developer to be concerned with the details of the integration process, such as the execution plan that has to be used to retrieve and integrate the data, or the search form submissions that have to be performed to answer a query. The key problem when dealing with (application or unified) search forms is that they have some query capabilities, which have to be taken into account when posing queries over it. Furthermore, the search forms are human-oriented and they have not formally-defined semantics.

In this paper, we propose IntegraWeb as a reference framework to perform deep-web data integration by means of high-level structured queries. IntegraWeb emerges as the synergy between the research efforts made on database integration and the Deep Web, and it uses the advances of both research fields in combination. To the best of our knowledge, there is not a proposal that uses high-level structured query languages to integrate deep-web data. Furthermore, we survey the state of the art in both research fields and how the existing techniques have to be used in our reference framework.

Database integration research field has focused on the integration of data stored in different applications. The existing techniques go from a unique entry point by which the user query the system (Mediators), to a totally distributed system

in which each application has its data model and the user query any of them (PDMS and PayGo systems). Also, the mappings between the different data models can be totally exact at the beginning of the integration process (Mediators and PDMS), or they can be basic and evolve during this process (PayGo systems).

The research on the Deep Web has focused on accessing and integrating web data. The main challenge on developing these techniques is to have into account the search forms, i.e., deep-web applications deliver their data by means of a search form that have to be filled in by the user. In this research field, there are two main approaches: retrieving and indexing data pages to allow search engines to have access to them (surfacing), or retrieving data pages, extracting data from these pages and aggregating the results, to deliver the data to the user as if only one deep-web application had been queried (virtual integration).

REFERENCES

[1] M. Álvarez, A. Pan, J. Raposo, F. Bellas, and F. Cacheda. Extracting lists of data records from semi-structured web pages. *Data Knowl. Eng.*, 64(2):491–509, 2008.

[2] M. Álvarez, J. Raposo, A. Pan, F. Cacheda, F. Bellas, and V. Carneiro. Crawling the Content Hidden Behind Web Forms. In *ICCSA*, pages 322–333, 2007.

[3] J. L. Arjona, R. Corchuelo, D. Ruiz, and M. Toro. From Wrapping to Knowledge. *IEEE Trans. Knowl. Data Eng.*, 19(2):310–323, 2007.

[4] M. K. Bergman. The Deep Web: Surfacing Hidden Value. *Journal of Electronic Publishing*, 7(1), 2001.

[5] P. A. Bernstein and S. Melnik. Model management 2.0: manipulating richer mappings. In *SIGMOD Conference*, pages 1–12, 2007.

[6] L. Blanco, V. Crescenzi, and P. Merialdo. Structure and Semantics of Data-IntensiveWeb Pages: An Experimental Study on their Relationships. *J. UCS*, 14(11):1877–1892, 2008.

[7] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon. XQuery 1.0: An XML Query Language. Technical report, W3C, 2007.

[8] M. E. Califf and R. J. Mooney. Bottom-Up Relational Learning of Pattern Matching Rules for Information Extraction. *Journal of Machine Learning Research*, 4:177–210, 2003.

[9] D. Calvanese, D. Lembo, and M. Lenzerini. Survey on methods for query rewriting and query answering using views. Technical Report D1.R5, Universitá di Roma, 2001.

[10] J. Caverlee and L. Liu. QA-Pagelet: Data Preparation Techniques for Large-Scale Data Analysis of the Deep Web. *IEEE Trans. Knowl. Data Eng.*, 17(9):1247–1262, 2005.

[11] S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[12] C.-H. Chang, M. Kayed, M. R. Girgis, and K. F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Trans. Knowl. Data Eng.*, 18(10):1411–1428, 2006.

[13] C.-H. Chang and S.-C. Kuo. OLERA: Semisupervised Web-Data Extraction with Visual Support. *IEEE Intelligent Systems*, 19(6):56–64, 2004.

[14] K. C.-C. Chang and H. Garcia-Molina. Approximate query mapping: Accounting for translation closeness. *VLDB J.*, 10(2-3):155–181, 2001.

[15] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, 33(3):61–70, 2004.

[16] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large Scale Integration: Building a MetaQuerier over Databases on the Web. In *CIDR*, pages 44–55, 2005.

[17] V. Crescenzi and G. Mecca. Grammars Have Exceptions. *Inf. Syst.*, 23(8):539–565, 1998.

[18] V. Crescenzi, G. Mecca, and P. Merialdo. RoadRunner: automatic data extraction from data-intensive web sites. In *SIGMOD Conference*, page 624, 2002.

[19] H. Davulcu, S. Vadrevu, and S. Nagarajan. OntoMiner: automated metadata and instance mining from news websites. *IJWGS*, 1(2):196–221, 2005.

[20] A. Doan and A. Y. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 26(1):83–94, 2005.

[21] J. Euzenat and P. Shvaiko. *Ontology matching*. Springer, 2007.

[22] M. J. Franklin, A. Y. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4):27–33, 2005.

[23] D. Freitag. Machine Learning for Information Extraction in Informal Domains. *Machine Learning*, 39(2/3):169–202, 2000.

[24] M. Friedman, A. Y. Levy, and T. D. Millstein. Navigational Plans For Data Integration. In *AAAI/IAAI*, pages 67–73, 1999.

[25] A. Gal, G. A. Modica, H. M. Jamil, and A. Eyal. Automatic Ontology Matching Using Application Semantics. *AI Magazine*, 26(1):21–32, 2005.

[26] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. D. Ullman, V. Vassalos, and J. Widom. The TSIMMIS Approach to Mediation: Data Models and Languages. *J. Intell. Inf. Syst.*, 8(2):117–132, 1997.

[27] L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio grows up: from research prototype to industrial tool. In *SIGMOD Conference*, pages 805–810, 2005.

[28] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[29] A. Y. Halevy, M. J. Franklin, and D. Maier. Principles of dataspace systems. In *PODS*, pages 1–9, 2006.

[30] A. Y. Halevy, Z. G. Ives, J. Madhavan, P. Mork, D. Suciu, and I. Tatarinov. The Piazza Peer Data Management System. *IEEE Trans. Knowl. Data Eng.*, 16(7):787–798, 2004.

[31] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management Systems. In *ICDE*, pages 505–516, 2003.

[32] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov. Schema mediation for large-scale semantic data sharing. *VLDB J.*, 14(1):68–83, 2005.

[33] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data Integration: The Teenage Years. In *VLDB*, pages 9–16, 2006.

[34] J. Hammer, J. McHugh, and H. Garcia-Molina. Semistructured Data: The Tsimmis Experience. In *ADBIS*, pages 1–8, 1997.

[35] B. He, T. Tao, and K. C.-C. Chang. Organizing structured Web sources by query schemas: a clustering approach. In *CIKM*, pages 22–31, 2004.

[36] H. He, W. Meng, Y. Lu, C. T. Yu, and Z. Wu. Towards Deeper Understanding of the Search Interfaces of the Deep Web. In *World Wide Web*, pages 133–155, 2007.

[37] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. *VLDB J.*, 13(3):256–273, 2004.

[38] A. Hogue and D. R. Karger. Thresher: automating the unwrapping of semantic content from the World Wide Web. In *WWW*, pages 86–95, 2005.

[39] O. Kaljuvee, O. Buyukkokten, H. Garcia-Molina, and A. Paepcke. Efficient Web form entry on PDAs. In *WWW*, pages 663–672, 2001.

[40] J. Kang and J. Choi. Recognising Informative Web Page Blocks Using Visual Segmentation for Efficient Information Extraction. *Journal of Universal Computer Science*, 14(11):1893–1910, 2008.

[41] N. Kushmerick. Learning to Invoke Web Forms. In *CoopIS/DOA/ODBASE*, pages 997–1013, 2003.

[42] A. H. F. Laender, B. A. Ribeiro-Neto, and A. S. da Silva. DEByE - Data Extraction By Example. *Data Knowl. Eng.*, 40(2):121–154, 2002.

[43] J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. F. Laender. Automatic generation of agents for collecting hidden Web pages for data extraction. *Data Knowl. Eng.*, 49(2):177–196, 2004.

[44] A. Langegger, W. Wöß, and M. Blöchl. A Semantic Web Middleware for Virtual Data Integration on the Web. In *ESWC*, pages 493–507, 2008.

[45] M. Lenzerini. Data Integration: A Theoretical Perspective. In *PODS*, pages 233–246, 2002.

[46] K. Lerman, L. Getoor, S. Minton, and C. A. Knoblock. Using the Structure of Web Sites for Automatic Segmentation of Tables. In *SIGMOD Conference*, pages 119–130, 2004.

[47] S. W. Liddle, D. W. Embley, D. T. Scott, and S. H. Yau. Extracting Data behind Web Forms. In *ER (Workshops)*, pages 402–413, 2002.

[48] L. Liu, C. Pu, and W. Han. XWRAP: An XML-Enabled Wrapper Construction System for Web Information Sources. In *ICDE*, pages 611–621, 2000.

[49] Y. Lu, H. He, H. Zhao, W. Meng, and C. T. Yu. Annotating Structured Data of the Deep Web. In *ICDE*, pages 376–385, 2007.
[50] J. Madhavan, L. Afanasiev, L. Antova, and A. Y. Halevy. Harnessing the Deep Web: Present and Future. In *CIDR*, page To be published, 2009.
[51] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Y. Halevy. Representing and Reasoning about Mappings between Domain Models. In *AAAI/IAAI*, pages 80–86, 2002.
[52] J. Madhavan, S. Cohen, X. L. Dong, A. Y. Halevy, S. R. Jeffery, D. Ko, and C. Yu. Web-Scale Data Integration: You can afford to Pay as You Go. In *CIDR*, pages 342–350, 2007.
[53] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Y. Halevy. Google's Deep Web crawl. *PVLDB*, 1(2):1241–1252, 2008.
[54] R. McCann, B. K. AlShebli, Q. Le, H. Nguyen, L. Vu, and A. Doan. Mapping Maintenance for Data Integration Systems. In *VLDB*, pages 1018–1030, 2005.
[55] G. A. Modica, A. Gal, and H. M. Jamil. The Use of Machine-Generated Ontologies in Dynamic Information Seeking. In *CoopIS*, pages 433–448, 2001.
[56] P. Montoto, A. Pan, J. Raposo, J. Losada, F. Bellas, and V. Carneiro. A Workflow Language for Web Automation. *Journal of Universal Computer Science*, 14(11):1838–1856, 2008.
[57] P. Montoto, A. Pan, J. Raposo, J. Losada, F. Bellas, and J. López. A Workflow-Based Approach for Creating Complex Web Wrappers. In *WISE*, pages 396–409, 2008.
[58] H. Nguyen, T. Nguyen, and J. Freire. Learning to Extract Form Labels. In *VLDB*, 2008.
[59] A. Pan, P. Montoto, A. Molano, M. Álvarez, J. Raposo, and Á. Viña. A Model for Advanced Query Capability Description in Mediator Systems. In *ICEIS*, pages 140–147, 2002.
[60] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Interactive query formulation over Web service-accessed sources. In *SIGMOD Conference*, pages 253–264, 2006.
[61] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical report, W3C, 2006.
[62] X. Qi and B. D. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2), 2009.
[63] B. Quilitz and U. Leser. Querying Distributed RDF Data Sources with SPARQL. In *ESWC*, pages 524–538, 2008.
[64] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. Technical Report 2000-36, Stanford University, 2000.
[65] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *VLDB*, pages 129–138, 2001.
[66] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.
[67] C. Rivero. From queries to search forms: an implementation. *IJCAT*, 33(4):264–270, 2008.
[68] A. D. Sarma, X. Dong, and A. Y. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD Conference*, pages 861–874, 2008.
[69] L. Shu, W. Meng, H. He, and C. T. Yu. Querying Capability Modeling and Construction of Deep Web Sources. In *WISE*, pages 13–25, 2007.
[70] J. Turmo, A. Ageno, and N. Català. Adaptive information extraction. *ACM Comput. Surv.*, 38(2), 2006.
[71] M. Vidal, A. S. da Silva, E. S. de Moura, and J. M. Cavalcanti. Structure-Based Crawling in the Hidden Web. *Journal of Universal Computer Science*, 14(11):1857–1876, 2008.
[72] J. Wang and F. H. Lochovsky. Data extraction and label assignment for web databases. In *WWW*, pages 187–196, 2003.
[73] M. J. Weal, H. Alani, S. Kim, P. H. Lewis, D. E. Millard, P. A. S. Sinclair, D. D. Roure, and N. R. Shadbolt. Ontologies as facilitators for repurposing web documents. *Int. J. Hum.-Comput. Stud.*, 65(6):537–562, 2007.
[74] Y. Zhai and B. Liu. Structured Data Extraction from the Web Based on Partial Tree Alignment. *IEEE Trans. Knowl. Data Eng.*, 18(12):1614–1628, 2006.
[75] Z. Zhang, B. He, and K. C.-C. Chang. Understanding Web Query Interfaces: Best-Effort Parsing with Hidden Syntax. In *SIGMOD Conference*, pages 107–118, 2004.
[76] Z. Zhang, B. He, and K. C.-C. Chang. Light-weight Domain-based Form Assistant: Querying Web Databases On the Fly. In *VLDB*, pages 97–108, 2005.
[77] H. Zhao, W. Meng, and C. T. Yu. Mining templates from search result records of search engines. In *KDD*, pages 884–893, 2007.

Using Degree of Impact on Software Artifacts to define priority for handling Inconsistencies

Dr. Randa Ali Numan Khaldi
Ajloun National Private University,
Ajloun, Jordan
randakhaldi@hotmail.com

Abstract - *In development process developers need to prioritize inconsistencies to identify the actions taken to handle inconsistency and to measure the impact of inconsistency-handling actions which is a key to effective action in the presence of inconsistencies. However, different stakeholders may assign different levels of priority to the same shared requirements from their own perspectives. The disagreement in the levels of priority assigned to the same shared requirements often puts developer in dilemma during the inconsistency handling. In this paper we define different algorithms for: measuring degree of impact on software artifacts; defining the ranges level using minimum and maximum values of degree of impact on software artifacts; identifying priority of handling and assessing the degree of risk and selecting handling actions depending on priority level defined whether it to resolve inconsistencies immediately or ameliorate or circumvent or ignore. These algorithms define levels of priority systematically to identify the different actions taken for handling inconsistency without depending on stakeholder perspective.*

Keywords: impact on software artifacts, handling inconsistency, identifying priority of handling, managing inconsistencies, level of priority.

1. Introduction

Definitely any changes done to a software requirements specification would create an inconsistency which we cannot avoid during development process. If we enforce consistency this means that the change has to be delayed until the problem is sorted out, during which the desired change cannot be represented.

It is often desirable to tolerate and even encourage inconsistency (Gabbay & Hunter, 1992) [2], to maximize design freedom, to prevent premature commitment to design decisions, and to ensure all views are taken into account.

Management of inconsistency consists of a number of activities (Nuseibeh, 1994a) [3]: -

- Detection of inconsistency which *focuses* on identifying specification knowledge that breaks a consistency rule.
- Classification of Inconsistency which focuses on identifying the kind of inconsistency that has been detected in a specification such as, the CONMAN system (Schwanke & Kaiser, 1988) [4] which identifies six different kinds of inconsistency that may arise in programming, and uses this classification to react accordingly.
- Handling inconsistency which *focuses* on acting in the presence of inconsistencies (Finkelstein, et al.,1994a) [5]. For example, when an inconsistency is detected, the appropriate action may be one of these actions:
  - **Ignore** - this is appropriate if the inconsistency is relatively isolated, and its presence does not prevent further development, or where the level of risk does not justify the cost of fixing it.
  - ***Delay*** - this may be used when further information is required, but is not immediately available.
  - **Circumvent** - in some cases it may be sensible to circumvent the inconsistency by disabling or modifying the rule that was broken, for example because it represents a specific exception to a general consistency rule.
  - **Ameliorate** - in many cases it will be possible to take steps that improve the situation, but which don't necessarily remove the inconsistency. This is an incremental resolution, which is appropriate where resolution involves a number of actions by different parties, and where only some of these actions can be taken immediately.
  - **Resolve** - to take actions that immediately repair the inconsistency. The actions may be as trivial as deleting elements of the specification that give rise to the inconsistency, or as complex as invoking a negotiation support tool to find a resolution.

So it's too important to measure the degree of impact for these inconsistencies on Software Artifacts even before deciding appropriate action for handling these inconsistencies.

2. Our Contribution

In previous paper (Randa Khaldi ,2010) [24] we define a new technique for Detecting and Locating Inconsistencies

in Software Requirements through locating inconsistency and building a matrix:

$$X_{ij} = \{\{x_{11}, x_{12}, ..., x_{1j}\}, ..., \{x_{i1}, x_{i2}, ..., xij\}\} \quad (1)$$

Where:

i - Number of requirements
j - Number of consistency rules
This matrix consists of (i) rows and (j) columns for example:

$$X_{ij} = \begin{array}{c} \\ r_1 \\ r_2 \\ . \\ r_i \end{array} \begin{array}{c} \begin{array}{cccc} R_1 & R_2 & & R_j \end{array} \\ \begin{bmatrix} 00 & 01 & ... & 10 \\ 01 & 00 & ... & 01 \\ ... & ... & ... & ... \\ 00 & 00 & ... & 00 \end{bmatrix} \end{array} \quad (2)$$

Where:
r*i* - Requirement with (i) number
R*j* – Consistency broken rule with (j) number

We use this matrix to find out for each requirements ri = {r1,r2,…,ri} how many consistency rules each requirement broke from the sets of consistency rules Rj= {R1,R2,…,Rj}.

2.1 Measuring the Degree of Impact on Software Artifacts.

We count the number of inconsistencies detected in each requirement (NXi) with the number of consistency rules broken in each requirement (NRj) and storing them in our Meta Data.

Algorithm (See Figure 1) starts with entering the number of inconsistencies in each requirement and the number of broken consistency rules in each requirement. According to these numbers we are checking the proposed impact on software artifact according to the following initial rules:

$$di = \sum NXi + \sum NRji \quad (3)$$

Where:
di - Degree of impact on software artifacts for the (i) requirement.
NXi - Number of inconsistencies in the (i) requirement.
NRji - Number of broken rules (Rj) in the (i) requirement.
i - Number of requirement.

- If (di) is less than the sum of the number of inconsistencies on the next requirement with the number of broken consistency rules in this requirement (di≤di+1) then we start to check if our di is less the dmax which we define it equal 100 if it is yes we keep the value of dmax the same and we write this di for this requirement else we change the value of dmax and put it equal di and we also write this di for this requirement and store it.
- If (di) is greater than the sum of the number of inconsistencies on the next requirement with the number of broken consistency rules in this requirement (di ≥ di+1). Then we start to check if our di is less than dmax : if it is yes then we perform another check. If our di ≥ dmin then we will keep the value of dmin the same and we write this di for this requirement, else we change the value of dmin and put it equal di also we write this di for this requirement. Then we are checking if the number of requirement (i) less than or equal the constant (n) which is equal to the number of requirements stored if is yes then we enter the next requirement ri = ri+1 else we stop.

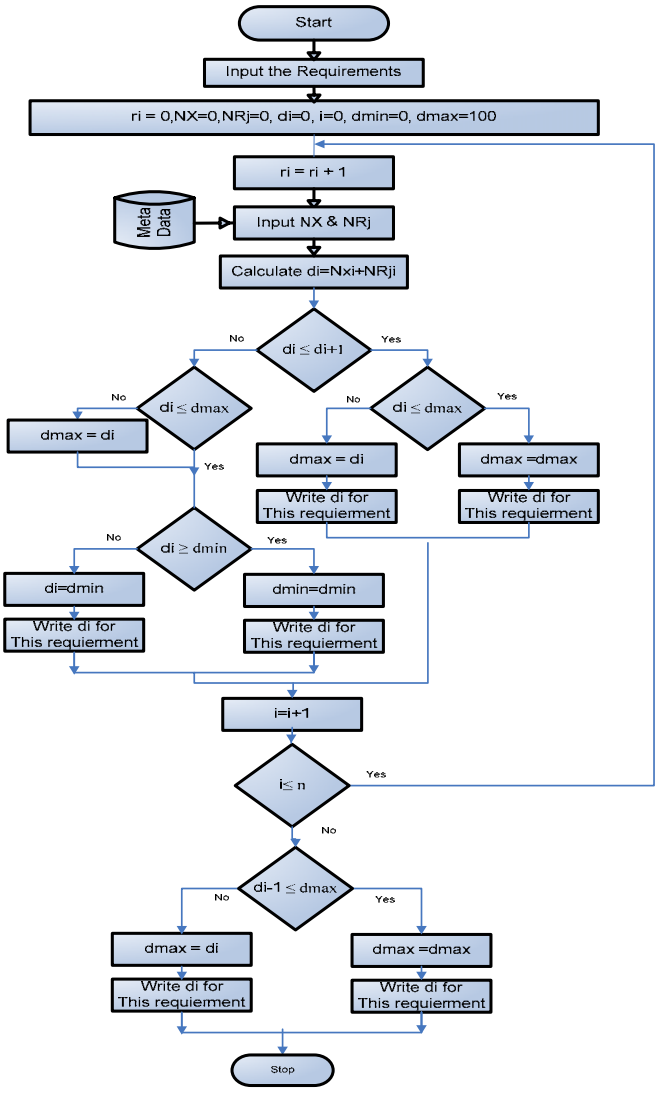


Figure 1: Algorithm for measuring degree of impact on software artifacts of inconsistencies

All these information and numbers are stored in our Meta Data for further use.

2.2 Identifying Priority of Handling

We use degree of impact on software artifacts (di) to define priority of taking handling action. Each requirement has a degree of impact of its inconsistencies and this degree of impact was counted and measured in algorithm. Each inconsistency was marked by adding additional information about the type of broken rules, the type of action that caused inconsistency, the degree of impact on software artifacts, the owner of the taken action in this requirement and its source. Also we stored data about the number of inconsistencies in each requirement specification.

We are defining the ranges of priority level according to the values of (di) (see Figure 2) the algorithm will be as follow:

- Input the value of dmax and dmin from Meta Rules and Data
- Check if the value of dmax is even then divide dmax by 4 ,

$$dmax / 4 = P \quad (4)$$

Where P is a constant which will be added further

- If the value of dmax is odd then add one to dmax and divide dmax by 4
- The first range of priority level P4 will be:

$$dmin \leq di \leq R \text{ , where } R = dmin + P \quad (5)$$

- The second range of priority level P3 will be:

$$R < di \leq R1 \text{ , where } R1 = R + P \quad (6)$$

- The third range of priority level P2 will be :

$$R1 < di \leq R2 \text{ , where } R2 = R1 + P \quad (7)$$

- The fourth range of priority level P1 will be :

$$R2 < di \leq dmax \quad (8)$$

Then we are identifying the priority of handling for each requirement with its inconsistencies as follow:

- If $dmin \leq di \leq R$ then these requirements with their inconsistencies are classified as group of priority level 4 which will be denoted as (P4).
- If $R < di \leq R1$ then these requirements with their inconsistencies are classified as group of priority level 3 which will be denoted as (P3).
- If $R1 < di \leq R2$ then these requirements with their inconsistencies are classified as group of priority level 2 which will be denoted as (P2).
- If $R2 < di \leq dmax$ then these requirements with their inconsistencies are classified as group of priority level 1 which will be denoted as (P1).

Our Algorithm (See Figure 3) starts by entering requirement and entering the calculated (di) in the algorithm then algorithm performs checking according to (di) and classifying them into groups of priority level which was defined above.

All these requirements with their inconsistencies and their group of priority level defined are stored in Meta Rules and Data for further use in assessing the degree of risk which will help us to select the handling action that must be taken.

In our algorithm we denote:

ri - Requirement with (i) number.

di - Degree of impact on software artifacts for (i) number.

i - Number of requirements.

n - Constant which is equal the number of requirement stored

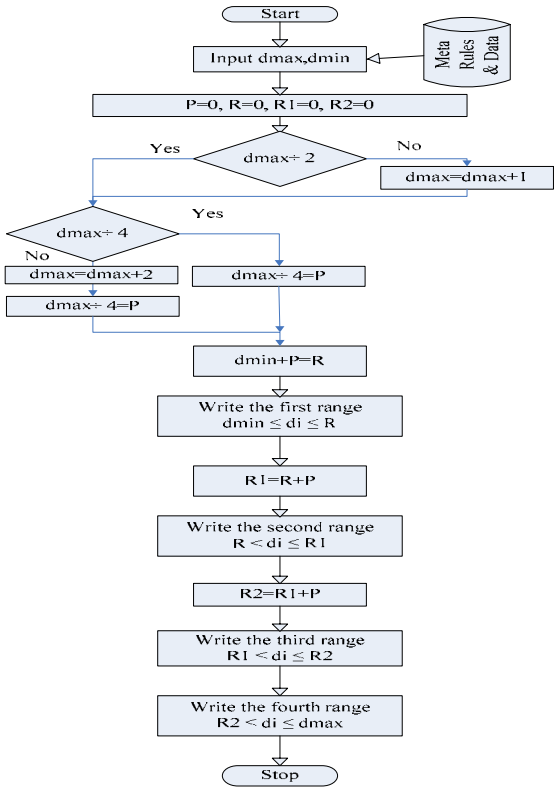


Figure 2: Algorithm for defining the ranges level using the maximum and minimum values of degree of impact on software artifacts (di)

2.3 Selecting Handling actions.

We define an algorithm for assessing the degree of risk on software artifacts according to its degree of impact on software artifacts and according to its level of priority.

For example: If inconsistencies in requirement ri are classified as group level (P1) which indicates that we have to deal with these inconsistencies immediately because they have the first priority, so these inconsistencies have a very high risk on software artifacts, which means that we have to select an immediate action to resolve these risky inconsistencies.

According to the above we define the following:

- If degree of impact on software artifact (di) has priority (P1) then classify this requirement (ri) with its inconsistencies in group High risk (H) and taken action will be to resolve inconsistencies.
- If degree of impact on software artifacts (di) has priority (P2) then classify this requirement (ri) with its

inconsistencies in group Moderate risk (M) and taken action will be to ameliorate inconsistencies.

- If degree of impact on software artifacts (di) has priority (P3) then classify this requirement (ri) with its inconsistencies in group Moderate risk (M) and taken action will be to circumvent inconsistencies.
- If degree of impact on software artifacts (di) has priority (P4) then classify this requirement (ri) with its inconsistencies in group Low risk (L) and taken action will be to ignore inconsistencies.

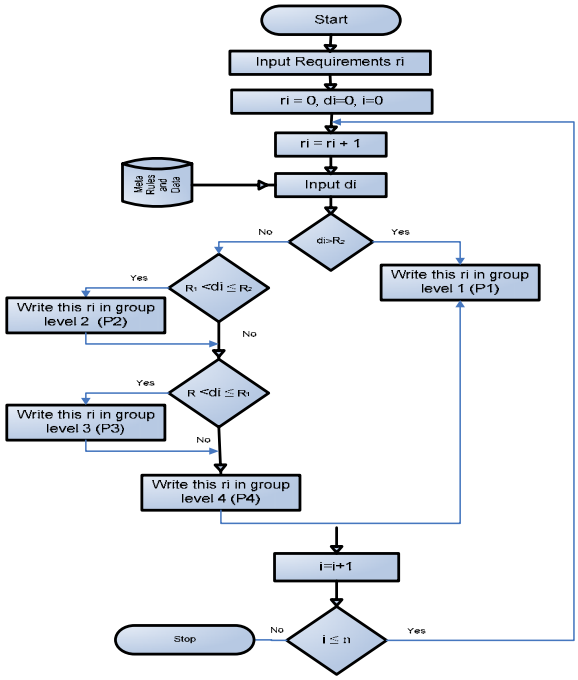


Figure 3: Algorithm for identifying priority of handling.

Our Algorithm (See Figure 4) starts when requirement (ri) with their inconsistencies entered. For each requirement we check the degree of impact on software artifacts (di) if this (di) is with priority level (P1) then classify it and write it in group of high risk (H) and write the value of degree of risk for this (ri) :

$$\text{drisk}\ i = di \qquad (9)$$

Where:

drisk i - degree of risk on software artifact for (ri) requirement.

di - degree of impact on software artifacts for (ri) requirement with (i) number which is equal sum number of inconsistencies and number of consistency broken rules in (ri) requirement.

This value of drisk i is stored in Meta Rules and Data and the selected taken action must be to *resolve inconsistencies immediately.*

Else we check if this (di) is with priority level (P2) then classify requirement (ri) with its inconsistencies and write it in group of Moderate risk (M) and let the drisk i = di and write this value of drisk i for this requirement (ri) in Meta Rules and Data and the selected taken action must be to *ameliorate inconsistencies.*

Else we check if this (di) is with priority level (P3) then classify requirement (ri) with its inconsistencies and write it in group of moderate risk (M) and let drisk i = di and write this value of drisk i for this requirement (ri) in Meta Rules and Data and the selected taken action must be to *circumvent inconsistencies.*

Else write this (di) is with priority level (P4) and classify it in group of Low risk (L) and let drisk i = di and store this value in Meta Rules and Data and the selected taken action must be to *ignore inconsistencies.*

In our Algorithm we denote:

ri - Requirement with (i) number.
i - Number of requirements.
n - Constant, which is equal the number of requirement stored.
P1 - Priority level one.
P2 - Priority level two.
P3 - Priority level three.
P4 - Priority level four.

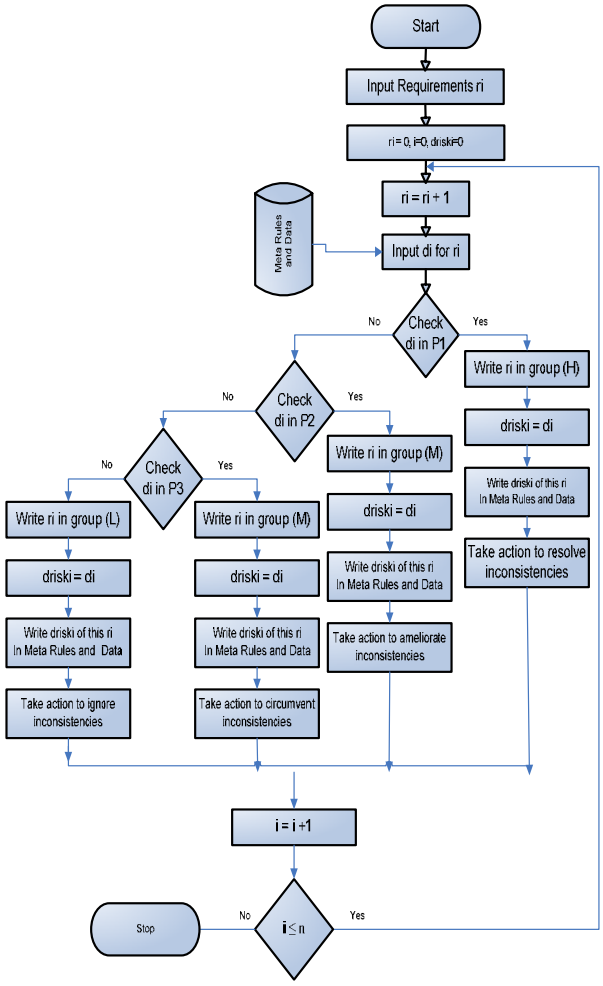


Figure 4: Algorithm for assessing the degree of risk and selecting handling actions

3. Summary and Future work

We defined a systematic approach for measuring the impact and the risk of inconsistencies on Software Artifacts, which will help solving many problems such as directing the process of development process and assisting the process of verification and validation. Also this approach will help in evaluating or estimating the consequences of handling actions; the frequency of inconsistencies or even the frequency of failure in the development process in order to assess the system reliability and the degree of progress in development process.

Future work is needed to measure the impact and the risk of inconsistencies on Software Artifacts after handling, to assess the Consequences of handling inconsistencies, Assess the Degree of Progress in development process. Also to build a knowledge base system to direct management of inconsistency process by building a machine learning system. The machine learning system will take use of all collected and stored information in Meta Data for automated reasoning; learning from previous experiences; statistics; pattern analysis and further data mining.

References:

[1] B. Nuseibeh, J. Kramer and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *IEEE Transactions on Software Engineering*, 20(10): 760-773, October 1994.

[2] D. Gabbay and A. Hunter; "Making Inconsistency Respectable: A Logical Framework for Inconsistency in Reasoning: Part 2"; (In) Symbolic and Quantitative Approaches to Reasoning and Uncertainty, LNCS, Springer- Verlag, pp.129-136, 1992.

[3] B. Nuseibeh, "Computer-Aided Inconsistency Management in Software Development", (Technical Report No. 95/4), Department of Computing, Empirical College, London, UK., 1994a.

[4] R. W. Schwanke and G. E. Kaiser; "Living With Inconsistency in Large Systems"; Proc. of the Int. Workshop on Software Version and Configuration Control, Grassau, Germany,

[5] A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh; "Inconsistency Handling in Multi-Perspective Specifications", IEEE Transactions on Software Engineering vol. 20, no.8, pp.569-578, 1994a.

pp. 98-118; B. G. Teubner, Stuttgart, , 27-29 January 1988.

[6] B. Nuseibeh, "To Be and Not To Be: On Managing Inconsistency in Software Development", Proceedings of 8th IEEE International Workshop on Software Specification and Design (IWSSD-8), Schloss Velen, Germany, pp. 164-169, 22-23 March 1996.

[7] R. Balzer; "Tolerating Inconsistency"; Proc. of 13th International Conference on Software Engineering (ICSE-13), IEEE CS Press, Austin, Texas, USA, , pp.158-165, 13-17th May 1991.

[8] G. Spanoidakis, A. Zisman; "Inconsistency Management in Software Engineering: Survey and Open research issues; Handbook of Software Engineering and Knowledge Engineering, (eds.) S. K. Chang, World Scientific Publishing Co., 2001.

[9] K. Narayanaswamy and N. Goldman ; "Lazy Consistency: A Basis for Cooperative Software Development"; Proc. of Int. Conf. on Computer-Supported Cooperative Work (CSCW '92), ACM SIGCHI & SIGOIS, Toronto, Ontario, Canada, 31st October - 4th, pp. 257-264, November 1992,.

[10] T. M. Hagensen and B. B. Kristensen ; "Consistency in Software System Development: Framework, Model, Techniques & Tools", *Software Engineering Notes (Proc. of ACM SIGSOFT Symposium on Software Development Environments)*, SIGSOFT & ACM Press, vol.17, no. 5, pp. 58-67, 9-11th December 1992.

[11] B. Nuseibeh; "A Multiple-Perspective Framework for Method Integration", PhD Thesis, Department of Computing, Imperial College, London, UK, October 1994b.

[12] V. Gervasi, D. Zowghi; Reasoning About Inconsistencies in Natural Language Requirement", ACM Transactions on Software Engineering and Methodology, vol. 14, No. 3, pp. 277-330, July 2005.

[13] M. L. G. Shaw, B. R. Gains;" Comparing conceptual structures: consensus, conflict, correspondence and contrast, Knowledge Acquisition: An International Journal, vol. 1, pp. 341-363, 1989.

[14] J. Iyer, D. Richards; "Evaluation Framework for Tools that Manage Requirements Inconsistency", Proceedings of the 9th Australian Workshop on Requirements Engineering (AWRE'04), 2004.

[15] B. Nuseibeh, ; "Towards a Framework for Managing Inconsistency Between Multiple Views, *Proceedings Viewpoints 96: International Workshop on Multi-Perspective Software Development*, ACM Press, pp. 184-186, 1996.

[16] C. Nentwich, W. Emmerich, A. Finkelstein and E. Ellmer; "Flexible Consistency Checking", ACM Transactions On Software Engineering and Methodology, vol.12, no. 1, pp. 28-63, 2003.

[17] A. Hunter ; "Evaluating the significance of inconsistency . In Proceedings of the International Joint Conference on AI (IJCAI'03), pp.468-473, 2003 .

[18] K.Knight ; "Two information measures for inconsistent sets. Journal of Logic, Language and Information,vol. 12, pp. 227-248, 2003.

[19] D. Avison and G. Fitzgerald; "Where Now for Development Methodologies", Communications of the ACM, 46(1), pp.79-81, Jan. 2003.

[20] C. Nentwich, W. Emmerich, A. Finkelstein and E.Ellmer ; "Consistency Management with Repair Actions", Proceedings of the 5th International Conference

on Software Engineering (ICSE-03), Portland, Oregon, IEEE CS Press, May 2003.
[21] L. Bertossi, A. Hunter and T. Schaub ; "Introduction to Inconsistency Tolerance", LNCS 3300, Springer-Verlag , Berlin Heidelberg, pp. 1-14, 2004.
[22] R. A. Khaldi, S. Abu-Soud; **"**A Systematic Approach for Managing Inconsistency in Software Requirements", Proceedings of the 1st International Conference on Digital Communications and Computer Applications (DCCA2007), Jordan, Amman, March 2007.
[23] Randa Khaldi , Dr. Saleh Abu-Soud," Inconsistency Management in Software Functional Requirements: A Machine Learning System", (MLMTA'07) proceedings, 2007,USA, Las Vegas, June 2007.
[24] Randa Khaldi, "A New Technique for Detecting and Locating Inconsistencies in Software Requirements", The International Arab Conference on Information Technology (ACIT'2010), University of Garyounis, Benghazi, Libya, December 14-16, 2010.
[25] S. Easterbrook;" Negotiation and the Role of the Requirements Specification", Appears in P. Quintas (ed.) "Social Dimensions of Systems Engineering: People, processes, policies and software development". London: Ellis Horwood, pp144-164, 1993.
[26] B. Nuseibeh, S. Easterbrook and A. Russo; "Leveraging Inconsistency in Software Development", IEEE Computer, vol.33, no. 4, pp. 24-29, April 2000.
[27] B. Nuseibeh, S. Easterbrook;" The Process of Inconsistency Management: A Framework for Understanding, Proceeding 1st International Workshop on the Requirements Engineering Process (REP'99), Sep. 1999.
[28] A. Hunter and B. Nuseibeh; " Managing Inconsistent Specifications: Reasoning, Analysis, and Action", Transactions on Software Engineering and Methodology, Oct. 1999.
[29] B. Nuseibeh, S. Easterbrook and A. Russo; "Making Inconsistency Respectable in Software Development", Journal of System and Software, vol. 58, no. 2, pp. 171-180, 2001.

Coverage-Based Test Sequences for FSM/EFSM Models

Ali Y. Duale
IBM, Poughkeepsie, NY

M. Umit Uyar
The City College of the City University of NY

Abstract:
Generation of minimum-length test sequences (MLTS) for FSM/EFSM models has been the focus of recent studies. Although it aims to optimize test cycles and expedites product marketing, an MLTS may fail to reveal potential discrepancies between an implementation and its specification. In this paper, MLTS methods combining FSM/EFSM models and pseudo random test generation techniques to enhance test coverage are presented.

Keywords: FSM, EFSM, minimum test sequences, random testing, test coverage, computer architecture.

1. INTRODUCTION

In general, test coverage measurement is both critical and challenging. Unfortunately, test coverage is often compromised due to tight deadlines, ambiguities in specification, and inefficiencies or lack of test tools. Typically test sequences are generated with the intention of reducing (minimizing if possible) the test length while covering as many aspects of the specification model as possible, mainly to avoid redundancies in tests, reduce development cycle and hence expedite marketing. If, however, the test length is reduced too much, the quality of the system may be negatively impacted, leading to failures at the field.

If a minimum-length test sequence (MLTS) fails to include the traversal of a set of paths that would create certain significant scenarios for the system operation, it is fair to assume that the test coverage is compromised. This paper presents algorithms designed to enhance test coverage for FSM/EFSM models. The presented algorithms combine techniques used for random test generations [1] and test generation for FSM/EFSM models.

2. BACKGROUND

FSM and EFSMs have been widely used to model a large variety of hardware and software systems such as communication protocols, digital logics and computer programs [2]. An FSM/EFSM model can be viewed as a directed graph [2] which consists of nodes

(representing the states), edges connecting the states (representing the state transitions), and input/output clauses associated with the edges (corresponding to the inputs to trigger these transitions and the expected outputs generated by the system). FSM/EFSM models considered in this paper are assumed to be strongly connected (each state can be reached from any other state) and deterministic (a given output produces a known output and causes state transition).

Test coverage is used to describe the degree to which an SUT has been tested or stressed. Although benchmarks can be used to qualify the capability of passing certain tests for a specific product, there is no general standard that can cover all functions of the SUT, especially when it is new in the industry.

In computer architecture verification, a large number of valid instructions can be used as the basis for the test cases. Ideally, one would assure that each instruction is tested for all possible combinations. This, however, would lead to an extremely large number of possibilities which, unfortunately, cannot be covered in a reasonable period of time for a product development cycle. Duale et al [11] presented methods to estimate and enhance the test coverage using pseudo random test generation [2,3,4,5,7,8]. Test cases were built by using streams of instructions and/or input/output commands selected from a pool of instructions. The authors created groups of instructions to represent a larger set of instructions. The selected groups are then monitored for their test coverage during and after test case building.

3. TEST COVERAGE ESTMATION AND IMPOROVEMENT FOR FSM/EFSM MODELS

In test generation for specifications modeled as FSMs or EFSMs, traversal of a given path may trigger a unique behavior (such as an arithmetic overflow) in a system under test (SUT). An MLTS that does not include edges that would create such desired outcomes lacks proper coverage. As an example, let us consider a path in the MLTS for the FSM presented in Fig. 1. In this example, edge e_0 is followed by edge e_2 which will never expose the behavior of the SUT when *X* overflows. In other cases, an MLTS may not simultaneously include more than one interesting/crucial test scenarios that need to be tested to cover potential corner cases. Fig. 2 shows a case where the SUT can exhibit both *X* and *Y* overflows. An MLTS that includes the edges of $e_2\ e_0\ e_3$ detects the Y overflow while *X* overflow can be tested by generating an MLTS with the edge sequence of $e_3\ e_1\ e_2$. Therefore, it is possible that there is no single MLTS that simultaneously covers those two cases.

This paper discusses means to empower the graph traversal of FSM/EFSM models with inbuilt intelligence such that optimal MLTS set(s) with the highest test coverage are generated. Test coverage techniques for random test generation techniques will be expanded to enhance test coverage for FSM/EFSM models. This paper introduces two different approaches that, in addition to the generation of MLTS, assure the traversal of interesting/crucial paths.

Let us now describe simple cases of these two approaches where a given set of MLTSs is required to be included in the test cases. The approaches can be expanded to more complex cases where specific intermediate final results are part of the test case coverage metrics. In both approaches the following abbreviations are used:

- MLTS = a set containing minimum test sequence
- GRMLTS = a set of distinct MLTS
- DMLTS = desired set of MLTS

3.1 Random MLTSs:

In this approach sets of MLTS are randomly generated until the predetermined paths are included in the test sequences. Whenever a new MLTS is randomly generated, it is checked if it includes the desired path(s). For example, suppose a path containing the edge sequence of e_3 e_1 e_2 is needed in the test sequence. MLTS sets are randomly generated until this desired edge sequence is picked. Randomly-generated MLTS can be achieved by randomly selecting an outgoing edge from a given state. This approach can be summarized as follows:

1. Generate a Random MLTS (RMLTS)
2. If current RMLTS is unique, add it to the Global GMLTS
3. Otherwise go to 1
4. Check if the GRMLTS includes DMLTS
5. If DMLTS is found – Test case Generation is done
6. Else Increment retry count
7. If retry count reaches maximum, exit with error
8. Else go to 1

This method may generate a number of MLTS before DMLTS is obtained. However, its simplicity makes it appealing. Tables 1 and 2 show two MTLS that are randomly generated. The MLTS shown in Table 1 will miss the creation of testing X overflow. On the other hand, the MLTS in Table 2 indicates a case where X overflows.

Edge Traversal	X
e_1	-
e_3	MAX
e_0	0
e_2	1

Table 1: MLTS that does not cause X to overflow.

Edge Traversal	
e_0	0
e_1	0
e_2	1
e_3	**MAX +1**

Table 2: MLTS that causes X to overflow.

3.2 Guiding Graph Traversal:

This approach requires selecting the next graph edge based on whether the edge would lead to the contribution of the desired path(s) in DMLTS. During the graph traversal, the outgoing edges of a given node are checked if they lead to obtaining the desired paths. The method requires that the paths that are built up until a node is visited should be stored. The selection of the next edge to be traversed at each node to obtain DMLTS is outlined as follows:

1. Randomly traverse the graph until reaching a state whose outgoing edge is e_f where e_f is the first edge of DMLTS.
2. From the outgoing edges of the current state, select an outgoing edge that would not break the DMLTS building.
3. Repeat 2 until DMLTS is obtained.

In the steps given above, each time a graph node is reached, the outgoing edge that is either part of the DMLTS or that will contribute to the construction of the DMLTS is selected. If all outgoing edges of a node are considered as contributing equally towards DMTLS, one of the outgoing edges from that node can be chosen arbitrarily.

This method becomes handy when specific the DMLTS is known but certain conditions are to be achieved. The creation of the DMLTS can be either set by the tester or can be based on recommendations from the product developers and customers. The goal is to set and assure the execution of certain combinations of input/output conditions. Table 3 shows a case where two concatenated MLSTs test cases where both X and Y overflow.

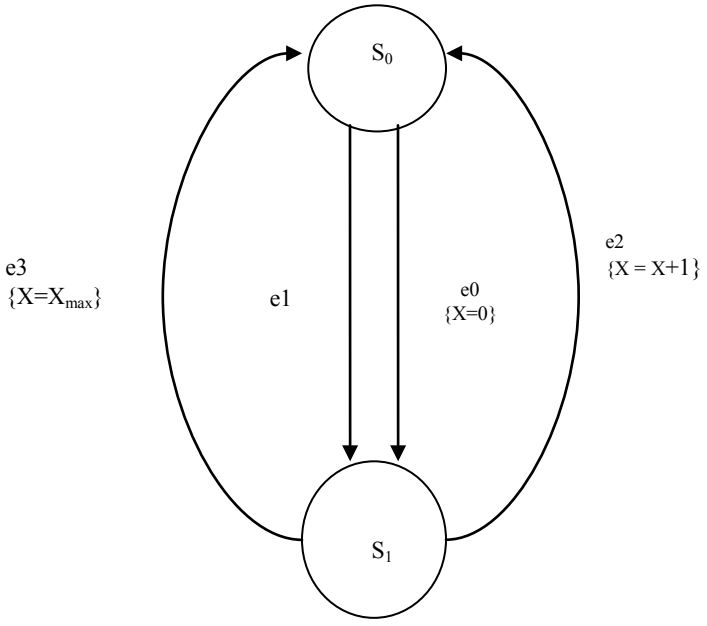


Fig 1: FSM with X Overflow.

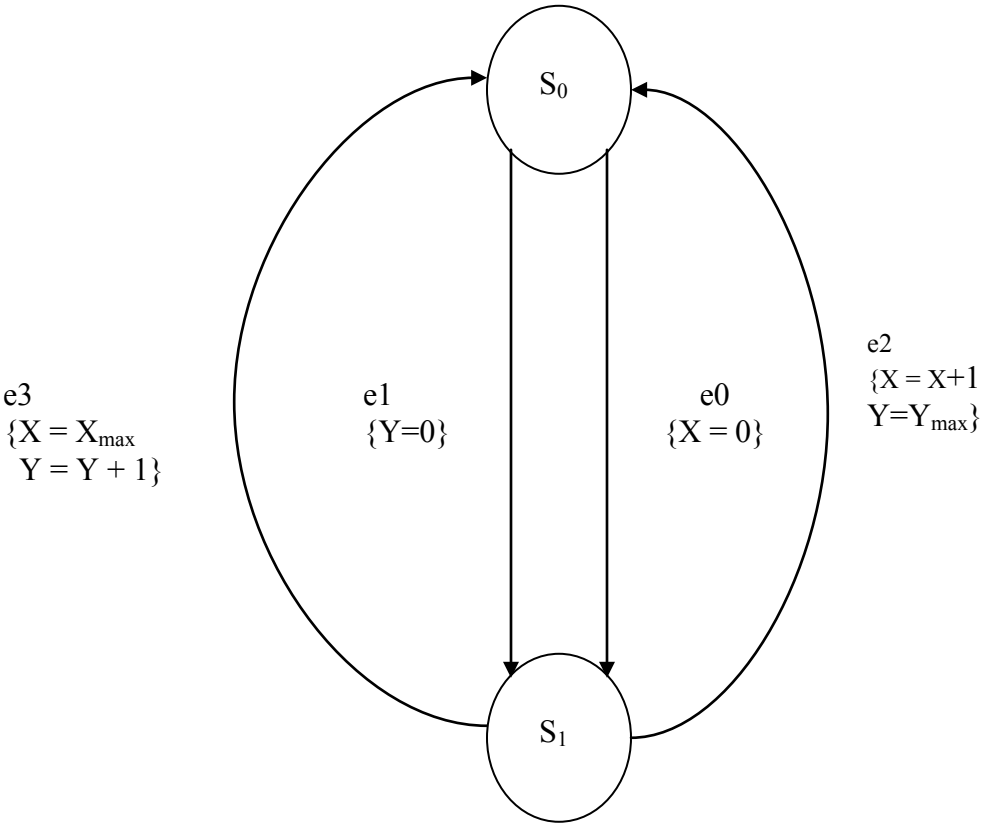


Fig 2: FSM with X and Y Overflow

Edge Traversal	X	Y
e_0	0	Initial
e_3	MAX	Initial + 1
e_1	MAX	0
e_2	**MAX + 1**	MAX
e_1	*MAX + 1*	0
e_2	*MAX + 2*	MAX
e_0	0	MAX
e_3	MAX	**MAX + 1**

Table 3: A set of two MLTSs causing both X and Y to overflow.

4. CONCLUSION

A method to enhance the test coverage for FSM/EFSM models is presented using pseudo-random test case generation and guided graph traversal approaches. Since the proposed method is scalable, it can be easily incorporated into the existing minimum-length test generation tools.

References

[1] M. Bailey, T. Moyers, S. Ntafos, An application of random software testing IEEE Military Communications Conference, 1995. MILCOM '95, Conference Record, vol. 3, 1995, pp. 1098-1102.

[2] B. Beizer, Software Testing Techniques, International Thomson Computer Press, 1990.

[3] A. Chandra, V. Iyengar, D. Jameson, R. Jawalekar, I Nair, B. Rosen, M. Mullen, J. Yoon, R. Armoni, D. Geist and Y. Wolfsthal, AVPGEN-A Test Generator for Architecture Verification, IEEE Tran. on VLSI Systems, vol. 3, no. 2, June 1995, pp. 188-200.

[4] T. Chen, Y. Yu, On the relationship between partition and random testing IEEE Trans. on Software Engineering, vol. 20 issue 12, Dec. 1994, pp. 977 –980.

[5] W. Debany, C. Hatmann, P. Varshaney and K. Mehrotra, Comparison of Random Test Vector Generation Strategies, IEEE International Conference on Computer-Aided Design (ICCAD-91 Digest of Technical Papers), 1991, pp. 244-247.

[6] Enterprise Systems Architecture/390 Principal of Operation, International Business Machine Corporation, 2002.

[7] M. Karam and G. Saucier, Functional Versus Random Test Generation for Controllers and Finite State Machines, Proc. Euro ASIC, 1992, pp. 207-212.

[8] S. Ntafos, On comparisons of random, partition, and proportional partition testing, IEEE Transactions on Software Engineering, vol. 27, issue 10, Oct. 2001, pp. 949-960.

[9] L. Fournier, Y. Arbetman, M. Levinger, Functional Verification Methodology for Microprocessors Using the Genesys Test-Program Generator, Pro Design Automation and Test, IEEE CS press, pp. 434-441, 1999.

[10] Ali Y. Duale, T. Bohizic, M. Decker, D. Wittig and G. Darling, Generation of Pseudo-Random Test Cases, Proc. 6th World Multiconference on Systemics, Cybernetics and Informatics, Orland, Fl, 2002, pp. 338-341.

[11] Ali Y. Duale, Theodore J. Bohizic, Dennis W. Wittig, Pseudo-Random System Testing: Coverage Estimation and Enhancement, Proceedings of the International Conference on Software Engineering Research and Practice, SERP 2005, Las Vegas, Nevada, USA, June 27-29, 2005, Volume 1.

Towards a Secure Service Oriented Product Line

Achour Ines[1], Sheima Khadouma[2], Lamia Labed[3] and Henda Ben Ghezala[1]
[1] Computer Science Department, Manouba University/ ENSI/ Lab. RIADI-GDL, Manouba, Tunisia
[2] Computer Science Department, Tunis University/ ISG/ Lab. SOIE and RIADI-GDL, Tunis, Tunisia
[3] Computer Science Department, Tunis University / ISG / Lab. RIADI-GDL, Manouba, Tunisia

Abstract - *In this work, we are dealing with service oriented applications which are based on service oriented architecture style. Also, we are interested on the large scale reuse paradigm which deals with a service line. This former envisions a family of similar service oriented applications. This approach promise gain in productivity and time to market. Combining SOA and service line is called SOPL (Service Oriented Product Line). The services which are autonomous play a fundamental role in SOPL. Assuring secure services is vital in establishing a climate of trust and confidence between Internet users and services providers. In this context, this paper deals with an attempt to extend the SOPL phases with security activities in order to produce secure service oriented applications as a result of a secure development process lifecycle.*

Key words: Service Oriented Architecture, Service Oriented Product Line, software security, secure domain engineering, secure application engineering.

1 Introduction

In a connected and open world, the confidence of Internet users is aligned to the provision of safe and secure services. We consider a family of service software applications that share common features where they differ from some others. This will enhance a better reuse and assure best productivity and time to market as promised by product line engineering in general. SOA architecture [1] provides applications with greater flexibility and a remedy to interoperability and heterogeneity problems. But thinking about family of such applications leads to a new concept known as SOPL. The security of the service oriented applications goes through integrating security requirements upstream (early in the software development process**).** Several studies [2, 3, 4] have shifted their interest from secure applications to the development process leading to their construction. Security concerns become proactive instead of reactive. In this paper, we propose to enrich the Service Oriented Product Line (SOPL) process by security activities in order to produce secure applications. Section 2 concerns a small outline of the software security. In Section 3, we elaborate an introduction to the Service Oriented Product Line process. In Section 4, we introduce the secure SOPL process. Section 5 deals with detailing the security activities in the domain engineering which is the first phase of the SOPL process. In Section 6, we highlight the security activities in the application engineering which is the second phase of the SOPL process. We then present in Section 7 a general discussion about related works and position our proposed approach. Finally, Section 8 summarizes our work and presents a discussion before outlining our long term future work.

2 Software Security

Security applications are added as features and mechanisms at the end of the development process independently of the other features of the system to protect the application against potential threats (reactive security). In contrast, secure applications are applications resulting from a development process in which most security problems are addressed and resolved in advance [5] (proactive security). They are designed and implemented so that they can function correctly in the presence of attacks. As we focus on the development of secure services, we cannot ignore efforts in strengthening the assurance of services security. Indeed, several studies in [6] and [1] have been developed such as WS-security [1], etc. But, this does not prevent us from noting that security is more robust if it is structured and integrated throughout the service development cycle.

3 SOPL Process

Service Oriented Architecture (SOA) is an approach that aids solving integration and interoperability problems [7]. Nevertheless, it does not provide support for systematic planned reuse as does Software Product Line engineering. This latter has the principles of variability, customization and systematic planned reuse and can be used to assist

SOA to reach its benefits. Hence, it appears a recent approach called Service Oriented Product Line (SOPL) which is introduced at the 11th edition of the International Conference SPLC (the 'Software Product Line Conference') in 2007.

The SOPL approach is a combination of Software Product Line engineering and Service Oriented Architecture approach providing thus solutions to many common software problems as reuse and interoperability issues [7]. It allows developing applications oriented SOA as Software Product Lines (SPL). Thus, the term Service-Oriented Product Line is used for service oriented applications that share commonalities and vary in an identifiable manner. Therefore, we deal with a service line which is a group of services sharing a common set of features.

In fact, SOPL process introduces variability management concepts into Service Oriented Architecture and applies SPL engineering life cycles as shown in figure 1:

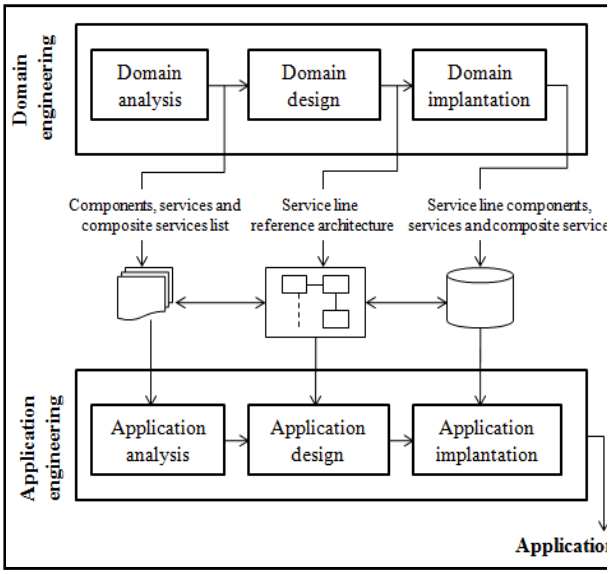


Figure 1: Service Oriented Product Line process

The domain engineering phase defines the development for reuse. The application engineering phase presents the development with reuse.

The SOPL process begins with a domain engineering phase [7]. It uses the feature model [8] and the business process model as inputs, in order to produce during the domain analysis a list of components, services and composite services candidates for reuse. Afterward, during the domain design, a variability analysis is executed through components and services identified previously. It defines the variability that will be implemented within the services and components. Subsequently, in an architecture specification activity, the components, services, and their flows will be specified using different views. During the application engineering [9], in the application analysis phase, components, services and composite services are selected from the list identified in the domain engineering. The selection responds to the functionalities required by the application to develop. Next, the configuration and the specialization of selected components, services and composite services, and architecture specification are performed during the application design. Product construction, in the application implantation phase, concludes the SOPL process.

4 Secure SOPL Process

In what follows, we present the life cycle of an SOPL based on [10, 11, 12, 13, 14], and essentially on Medeiros and al [7] and Helali [9] works to which we integrate security activities. As depicted in figure 2, these activities strengthen the security throughout the development process. We have inspired by security activities proposed by three validated secure development processes which are Touchpoints, CLASP and SDL [15, 2, 3, 4]. These processes are the most used, validated and proven in this field. Our secure SOPL inherits from the classic SOPL its two principle phases which are the domain engineering and the application engineering phases.

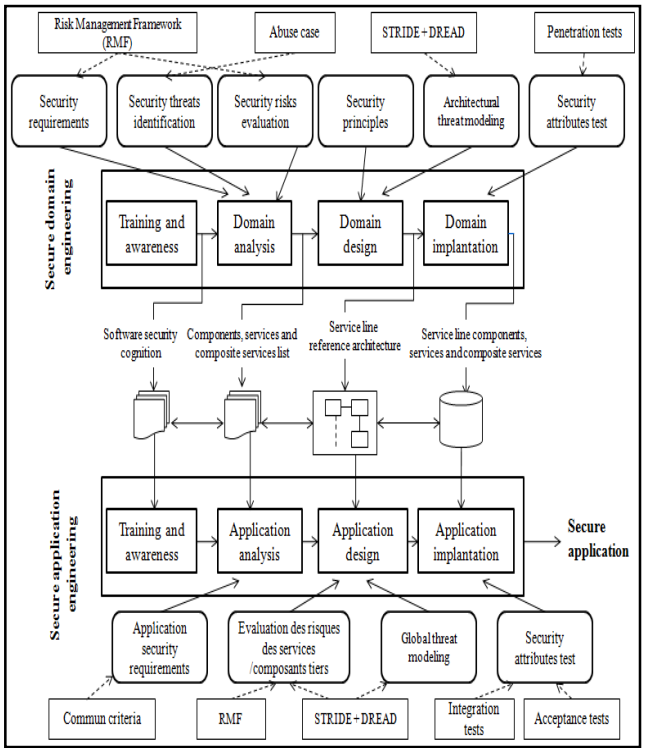


Figure 2: Secure Service Oriented Product Line process

5 Secure domain engineering

Secure domain engineering is divided into four phases namely: training and awareness, domain analysis, domain design and domain implantation.

5.1 Training and awareness

Touchpoints, CLASP and SDL processes emphasize the importance of training and education in security software to every member of the project. Thus, during this phase, the team member should be given a baseline education in software security in order to get knowledge in security engineering basics. It increases their awareness of the importance of the domain problems on which they operate and its broad scope. In this sense, it is essential to:

- Plan regular courses for the development team [4]. These courses cover the latest security software issues since it is an evolving field where it frequently reflects the emergence of new threats.
- Promote sharing and communication artifacts such as security threat models within the development team [3].
- Assign a security adviser to the project. This adviser helps developers with security related issues and possibly serves as a gateway between developers and a dedicated security team (if available) [3].

5.2 Domain analysis

For the domain analysis, we begin first by identifying and analyzing business requirements as in classical product line approaches [16]. We then conduct functional security requirements of the domain more deeply than in product line approaches. Then, we move to identify components, services and composite services candidates for reuse to analyze their variability thereafter. Finally, we conduct a risk management for each element of the reuse candidate list. Note that we adopt in this phase, the Risk Management Framework (RMF) proposed by the Touchpoints approach [2]. This choice is motivated by the robustness offered by this process in terms of risk management. But this doesn't exclude the use of other risk management techniques which are numerous in the security domain field.

5.2.1 Functional requirements identification

We apply, at this stage, the first activity of the RMF process called the business context understanding [2] to which we integrate activities related to SOPL. This activity helps to better assimilate functionalities and the domain of study by the developers community for reuse. It also presents the starting point for other activities measurement and risk analysis. This step includes the following activities:

- Resources and artifacts gathering [2]: it comes to identify resources and possible artifacts of the service line that we intend to develop (such as data used, etc.).
- Business processes representation: these processes are used to identify candidate services for reuse [7].
- Feature Model development by the application of the Feature Oriented Domain Analysis method (FODA) [8]: this model is used to identify candidate components for reuse.

5.2.2 Security requirements identification

The security requirements elicitation is performed by using a threat modeling and the commercial and political requirements related to the environment. We note that the threat modeling gives rise to a threat model document for the members of the development team helping them understand eventual threats and how to treat them. Threat modeling is an iterative process [17] spanning the entire cycle of our secure SOPL. This is explained by the impossibility of identifying all threats at once and by the dynamic aspect of applications faced to the need for an adaptation to changes. Threat modeling is a fundamental task for any security-oriented effort. In order to reinforce this point, we opt for several approaches to perform the threat modeling [3, 18]. During use case driven threat modeling, attacks to all functional use cases of the domain are identified. Resource driven threat modeling focuses on legal use of resources. Finally, through knowledge driven threat modeling, known attacks are assessed whether they can be valid and useful for the domain at hand. Apart from the aforementioned threat modeling, extra security requirements can be specified based on laws and regulations, contractual obligations and commercial considerations [2]. Before proceeding to the next step, developers must not forget to enrich the Feature Model with results obtained through abuse cases.

5.2.3 Components, services and composite services identification

This stage is based on the Feature Model and Business Process Models already established for determining the list of components, services and composite services that support business processes.

5.2.4 Variability analysis

The variability analysis activity starts with an analysis of similarities and variabilities between components and services with the purpose to reduce the number of service candidates. The similarity analysis consists in comparing the functionalities of components and services in order to join similar services with the fine-grained variability [7]. Indeed, variability is the ability to change or adapt software systems. Several mechanisms of variability management are available in the literature [19] such as parameterization, inheritance, conditional compilation, etc.

5.2.5 Risk management

Once the components, services and composite services list is established, we apply at this stage the risk management process RMF. Thus for each component, service and composite service, we proceed to:

- Business and technical risks Identification: a component, a service or a composite service is based on two aspects. First, a business aspect represents the purpose and the role of a component, a service or a composite service in relation to the global business context. Second, a technical aspect evokes the technical methods used for an element construction. So during this process activity [2], we identify business risks and technical risks that could have a negative impact on the items above. Thus, the analyst must begin by developing risk questionnaires [2] to ask about risks threatening the domain. These questionnaires are sent to the entire team of developers for reuse. Then, the analyst, based on results of questionnaires, focuses on identifying business objectives, business risks while trying to identify indicators of the latter, their occurrence probability, their impact, their cost, etc [2]. Then, the analyst proceeds to identify the technical risks (risk indicators, their probability of occurrence, their impact and severity level) [2].
- The synthesis and prioritization of risks: to better understand and manage a particular risk, the analyst is required to establish relationships between business requirements, business risks, and technical risks of the component or the service to be secured in a Goal-to-risk relationship table [2]. The prioritization process must take into account which business goals are the most important to the organization and which goals are the most threatened.

5.3 Domain design

For the domain design, we start with the security principles definition which will be adopted in the construction of the reference architecture. Then, we specify the architecture and finally we perform an architectural threat modeling.

5.3.1 Security principles definition

The robustness of a system proceeds imperatively by security choices undertaken during the architecture specification. Thus, according to [20] several security principles can be considered such as the assumption that any interaction with a third-party product is risky and so by considering incoming data as suspect and by isolating critical assets, etc.

5.3.2 Reference architecture specification

The reference architecture of the service line is built following security principles and architectural decisions taken at the variability analysis. During the architecture specification, different architectural views can be produced [7]:

- A structural view for representing the static structure of the architecture specified.
- A layer view for representing components and services organized in their layers.
- An interaction view showing communications and interactions between components and services when achieving a particular functionality.
- A dependency view showing the dependence information among services and components.
- A concurrency view showing parallel communication among services and components.
- A physical view showing the communication protocols and distribution of components and services.

5.3.3 Architectural threat modeling

During architectural threat modeling, it is essential to pursue threats identification started in the domain analysis. So, it is recommended to:

- Identify threats using the STRIDE method [18] which provides a classification of types of threats.
- Create attacks models that describe the attacker target, the attacker motivation and the used techniques [2].
- Evaluate threats identified using the DREAD model [18].
- Define the risk mitigation strategy: given a set of risks and their priorities, the next stage is to create a coherent strategy for mitigating (remove the feature, solving the problem, etc.) [2]. The risks mitigation takes into account cost, implementation time, likelihood of success, and impact over the entire corpus of risks.

5.4 Domain implantation

Domain implantation consists in: 1) the implementation of identified components, services and composite services and in 2) the verification of their adherence to specified business requirements and functional security requirements.

5.4.1 Components, services and composite services implementation

During this stage, developers for reuse build components, services and composite services of the asset base. They should respect a set of practices [21] to improve the security code such as: choosing a secure programming language (e.g J2EE [22]), using security standards dedicated to services (e.g WS-

Security [1]), using code analysis tools [23], etc. Also, developers for reuse are expected to prepare documentation to the development community with reuse by defining security software practices used during the secure engineering domain.

5.4.2 Components, services and composite services testing

After finishing the coding stage, the development team for reuse focuses on components, services and composite services testing. We suggest, therefore, applying two different approaches for testing:

- White-hat approach [15]: this approach allows testing business functionalities of components, services and composite services such as "Request for birth certificate" service in the case of an administration on line.
- Black-hat approach [15]: this approach allows testing security features and the resistance to possible attacks discovered during the risk analysis. Black-hat approach takes into account the worst case scenario of use cases. In this case, we suggest performing penetration tests [24].

Testing-related security activities should be documented and made available for the development team with reuse. Such documentation can be used for instance as a reference point for corrective actions oriented security.

6 Secure application engineering

Secure domain engineering produces reusable artefacts such as secure services and secure reference architecture. Now, for a given application, secure application engineering will instantiate and specialize the secure reusable assets. Note that new applications security issues emerge through assembling application's components, services and composite services. Hence, they can lead to security risks and damage the application. Secure application engineering is divided into four phases: training and awareness, application requirements analysis, application design and application implantation.

We are not going to present all of them because of space constrains.

6.1 Application requirements analysis

The application development team identifies, first, security requirements of the application derived from the service line. These security requirements are elicitated from use cases and the operational environment. Application security requirements are specified at the host level and the network level. Then, the security requirements are prioritized based on threat levels such as protecting sensitive data, features that require privileges, etc. At this level, we propose to apply Common Criteria method (Common Criteria for Information Technology Security Evaluation) [25]. This method presents functional requirements in the form of a components catalog to meet certain security objectives, and assurance requirements which provide actions to ensure compliance with these objectives. Based on this analysis, components, services and composite services are selected from the asset base built in the secure domain engineering.

6.2 Application design

The application design includes the configuration and the specialization of candidates for reuse previously selected, a risk assessment of third-party products (if available) and a threat modeling by identifying appropriate defense mechanisms.

After selecting components, services and composite services, the developers team may use third-party components or services (external to the asset base). This integration can lead to security risks that could damage the application security [15]. Developers can apply the RMF process to assess such risks. We note that this step is not really mandatory, but it may be possible.

The architecture specification is performed by instantiating the reference architecture and that by keeping only components, services and composite services required by the derived application. An audit of the consistency constraints set at the secure domain engineering should be conducted to ensure the compliance of the application architecture with the reference architecture.

It is appropriate to check with the threat model built during the application design phase the possible emergence of new threats that may arise through assembling components, services and composite services. This is explained by the fact that the threat modeling in the secure domain engineering is conducted for each component, service and composite service. It refers to a local threat modeling. However, the threat modeling in this phase must be performed on the whole set of components, services and composite services required by the derived application from the service line. It refers to a global threat modeling. In case of identification of a new threat, this latter is categorized as STRIDE categories [18] and developers must develop appropriate security strategies.

6.3 Application implantation

For the application implantation, developers begin by constructing the application. Then, they proceed to test its security features and its resistance to attacks and finally they elaborate the appropriate documentation.

To validate the developed application, we recommend to developers to perform integration tests [26] and acceptance tests [27]. Integration tests allow validating that components, services and composite services developed independently communicate together coherently. These tests can be automated by tools such as Eggplant [28] and JUnit [29], etc. Regarding acceptance tests, they allow testing the application in its real environment by the end user. The client in this case may provide the development team with a feedback from performed tests.

After ensuring that the developed application meets the specified business requirements and the functional security requirements, the development team with reuse produces the documentation and guides for security administrators and end users.

7 Discussion

Several works have been conducted for securing web services [1,6] such as WS-security, WS-Trust, WS-Conversation [1], etc. But, security is more robust if it is structured and integrated throughout the service oriented application development cycle. The SDL, CLASP and the Touchpoints approaches [4, 3, 2, 15], which are the most used, proved and validated approaches for producing secure applications can be used for service oriented applications. But, we want to use the systematic large scale reuse paradigm which causes the creation of another secure development process. For that, we have added secure activities in the SOPL process and we have inspired by those of the three mentioned validated processes. So, the resulted process is theoretically validated. It appears to be too high level and very descriptive because we need to add all the security activities. But when conducting an example to illustrate the use of the secure SOPL process, the process usage will be more concrete with more details. We in fact, apply it to a family of e-vote applications where security issues are numerous. We have identified several reusable services and components for different e-vote systems such as presidential election, laws elections, etc. In fact, similarities and differences appear in the e-vote systems. The secure domain engineering phase permits to obtain a secure reference architecture where the security attributes were deeply taken into account and produces several secure artifacts such as secure services. The secure application engineering phase allows obtaining different secure e-vote applications. The limitation of our proposed approach is that dealing with security issues involves the use of a lot of security techniques and methods which can be perceived by the user as difficult and boring but this is the price to pay for obtaining secure service oriented applications.

8 Conclusion

The principle aim of this work is to integrate security activities in the Service Oriented Product Line process in order to produce secure services proactively. In this work, we have inspired by the SDL, CLASP and the Touchpoints approaches [4, 3, 2, 15], which are the most used, proved and validated approaches for producing secure applications. So, we have added several security activities in the SOPL process and also advocate the use of different security methods, concepts and frameworks (such as RMF, STRIDE and Common Criteria) which are well suited for such context. This does not exclude the use of other security mechanisms which are numerous in the security field. This work aims to ensure the development of an application (service based in our case) by taking advantages of three concepts contributions: a large-scale reuse system that's product line engineering, service oriented architecture and software security. Our perspectives are first the application and the use of our secure SOPL in different E-Government domain applications, where we adopt SOPL reference architecture for a family of E-Government services. As mentioned previously, we have begun with a family of e-vote applications. Second, we would like to validate our proposed secure SOPL process and its use in different contexts such as e-commerce, e-learning, etc.

9 Acknowledgment

This work has been supported by the Tunisian project S2EG (Secured Systems for the E-Government) which presents the collaboration of three research structures: SOIE, CRISTAL and RIADI and financed by the ministry of communication technologies of Tunisia.

10 References

[1] Ort E., "Service-Oriented Architecture and Web Services: Concepts, Technologies, and Tools", Technical Report, SUN, 2005.

[2] MacGraw G., "Software Security: Building Security In", Addison-Wesley Professional, ISBN: 978-0-321-35670-3, 2006.

[3] OWASP Corporation, "CLASP Comprehensive Lightweight Application Security Process", 2006.

[4] Lipner S., "The trustworthy computing security development lifecycle", Computer Security Applications Conference, 2004. 20th Annual Publication, ISSN: 1063-9527, ISBN: 0-7695-2252-1, pp. 2-13, 2004.

[5] Goertzel K., Winograd T., McKinley H., Oh L., Colon M., McGibbon T., Fedchak E. and Vienneau R., "Software Security Assurance", State-of-the-Art Report (SOAR), 2007.

[6] Guruge A., "Web Services: Theory and Practices", Digital Press, 2004.

[7] Medeiros F., Romero S. and Santana E., "Towards an Approach for Service-Oriented Product Line Architectures", 13th International Software Product Line Conference (SPLC 2009), San Fransisco, CA, USA, 2009.

[8] Kang K., Cohen S., Hess J., Novak W. and Peterson S., "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[9] Helali R., "Product Lines approach for the derivation of software applications in E-Government", Master, university of Tunis, 2010.

[10] Berger T. and Gunther S., "Service-Oriented Product Lines: Towards a Development Process and Feature Management Model for Web Services", 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland, 2008.

[11] Heferich A., Herzwurm G. and Jess S., "Software Product Lines and Service-Oriented Architecture: A Systematic Comparison of Two Concepts", Proc. of the First Workshop on Service-Oriented Architectures and Software Product Lines, pp. 31-37, 2008.

[12] Lee J., Kim M., Muthig D., Naab M. and Park S., "Identifying and Specifying Reusable Services of Service Centric Systems Through Product Line Technology", Proc. of the First Workshop on Service-Oriented Architectures and Software Product Lines, pp. 57-67, 2008.

[13] Mannisto T., Myllarniemi V. and Raatikainen M., "Comparison of service and Software Product Family Modeling", Proc. of the First Workshop on Service-Oriented Architectures and Software Product Lines, pp. 47-57, 2008.

[14] Wienands C., "Synergies between Service-Oriented Architecture and Software Product Lines". Siemens Corporate Research Princeton, NJ, 2006.

[15] De Win B., Scandariato R., Buyens K., Grégoire J. and Joosen W., "On the secure software development process: CLASP, SDL and Touchpoints compared", Information and Software Technology, Vol.51, No. 7, pp. 1152-1171, 2009.

[16] Clements P., and Northrop L., "Software product lines: practices and patterns". Addison- Wesley, Boston, MA, 2001.

[17] Meier J.D., Mackman A. and Wastell B., "Patterns & practices Library", Microsoft Corporation , 2005.

[18] Meier J.D., Mackman A., Vasireddy S., Dunner M., Escamilla R. and Murukan A., "Improving Web Application Security: Threats and Countermeasures", Microsoft Corporation, 2003.

[19] Gomaa H., "Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures", Addison-Wesley Professional, ISBN:0201775956, 2004.

[20] Stoneburner G., Hayden C. and Feringa A., "Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A", Recommendations of the National Institute of Standards and Technology, 2004.

[21] Howard M. and Microsoft Corporation, "Fundamental practices for secure software development", Stacy Simpson, SAFECode, 2008.

[22] Batchelli A., "A Brief Introduction to J2EE", 2006, http://www.voneicken.com/courses/ucsb-cs290f-fa06/images/d/df/Lecture17.pdf, [12 Feb 2011].

[23] Security Innovation Corporation, "Hacker Report: Static Analysis Tools", 2004.

[24] AppLabs, "Web Application Penetration Testing", 2009.

[25] Common Criteria Recognition Agreement, "Common Criteria for Information Technology Security Evaluation", 2009.

[26] Bradley T., "Integration Testing", 2008.

[27] **"**Software Testing - Acceptance Testing", http://www.buzzle.com/articles/software-testing-acceptance-testing.html, [26 Feb 2011].

[28]http://www.testplant.com/products/eggplant_functional_tester, [26 Feb 2011].

[29] http://www.JUnit.org, [26 Feb 2011].

Cooperative Object-Oriented Programming in Python

A. Chan[1]
[1]Department of Mathematics and Computer Science
Fayetteville State University
North Carolina, 28301

Abstract - *In this article we will investigate the object-oriented features of the Python programming language. Python supports encapsulation with limited information hiding; it has full support in polymorphism and inheritance. Therefore, Python fits the definition of object-oriented programming languages. On the other hand, there are some "add-on" features that are commonly available in many main-stream object-oriented languages but are missing in Python. We will examine these features and, whenever possible, provide "work-around" so the user can enjoy the benefits of these missing features. We call this "cooperative object-oriented programming" because when we "simulate" these features, we often need to advise the users not to trespass into the forbidden zone so that the "work-around" can function as expected.*

Keywords: Python, Object-Oriented, Cooperative.

1 Introduction

Python is often advertised as a multi-paradigm programming language. It supports the imperative, object-oriented, and functional paradigms. But many programmers like to refer it as an object-oriented programming language.

According to Tucker and Noonan [10], a programming language is object-oriented if it supports an encapsulation mechanism with information hiding for defining abstract data types, virtual methods, and inheritance. Based on this definition, Python only barely fits into the object-oriented paradigm. It also lacks many features that are commonly available in other main-stream object-oriented languages. Although this does not hurt its status as an object-oriented language, it will make the language less pleasant to use compared to these main-stream object-oriented languages. In order to simulate these missing features, the programmer may have to depend on documentation and assume the users will follow the advice. Therefore, we came up with the term "cooperative object-oriented programming".

In this article, we first examine the features that qualify Python as an object-oriented programming language and discuss its strengths and weakness. We then look at the common features that appear in other object-oriented languages but are absent in Python. If possible, we will suggest techniques to simulate these features.

In Section 2, we discuss some previous work on "cooperative object-oriented programming". Section 3 presents the object-oriented features that are available in Python. Section 4 discusses the features that are commonly available in object-oriented programming languages but are missing in the Python language. We conclude our discussion in Section 5.

2 Previous Works

We are not the first to use the term "cooperative object-oriented programming." The term was first used by Nascimento and Dollimore [4], who talked about how to have a team of programmers cooperatively and efficiently develop object-oriented software using a pre-defined framework. Damm, Hansen, and Thomsen [2] used the same term to refer to the fact that we usually use the object-oriented paradigm to develop large-scale software, thus requiring a large team of programmers to work cooperatively. Silberztein [8] used this term in a slightly different context, in which he created an architecture that consists a large number of object-oriented components cooperating together to perform a required operation (in his case that is natural language processing).

We, however, use the term differently. In this paper, we use "cooperative object-oriented programming" to refer to the fact that a programming language is lacking some useful features that are usually available in other programming languages. In order to simulate these useful features, sometimes our only option is to document the missing features and ask the users not to touch the forbidden zone. Therefore, when we use the term "cooperative object-oriented programming," we are referring to the necessary cooperation between the users (the programmers) and the library (author).

3 Python's Object-Oriented Features

3.1 Encapsulation with Information Hiding

Encapsulation means putting things together as a single unit. This is not a new concept. Almost all programming languages provide some form of encapsulation, in the form of subroutines (procedures) and functions, which encapsulate the behaviors; and records

(structures) which encapsulate the data. What is unique in object-oriented programming is the encapsulation of behaviors and data into a single unit, which usually appears in the form of a class. We usually call the encapsulated data attributes and the encapsulated behavior methods. When a method is invoked, a message is sent to the object (and the object will respond to the message by executing the behavior encapsulated in the method). There is a distinction between a method and a message. A message is the flowing of execution between the sender (caller) and the receiver (the object), while a method is the implementation of instruction to the system on how to respond to the message.

Information hiding, on the other hand, is a new concept in object-oriented programming. The purpose of information hiding is to solve a common problem – the programmers' abuse of library code. It is a common phenomenon that if the programmers have access to some internal states/methods of a class, eventually some programmers will access those internal states/methods, either intentionally or unintentionally. The abuse makes maintenance of the code much more difficult than necessary. Therefore, it is often necessary to hide the internal working of a class from the users. The mechanism for information hiding is visibility. Most languages provide at least two levels of visibility (e.g. public and private in Smalltalk), some provide more (for example, public, private, protected, and package in Java). Python, based on its design, does not support visibility, and hence no information hiding. The Python standard suggests that names starting with underscores should be treated as private, and therefore users should not touch these names. However, there is no mechanism to actually prevent users from accessing "private" names. Python 3.0 uses name mangling to make accessing these "private" names more complicated, but it is still possible to access these "private" names, although it may require using some less-obvious methods. In fact, the Python Tutorial [7] claims that this "can even be useful in special circumstances, such as in the debugger."

Therefore, information hiding in Python depends on advisement; that is, advising the users not to touch the "private" names; and hoping that the users will follow the advice. This makes Python very suitable for small scale, single-person projects; however, it may be difficult to manage large scale, multi-programmer projects.

3.2 Virtual Methods

Virtual methods (also known as polymorphism) are free in Python. Names in Python do not need to be declared to be of specific types; in fact, the same name can be rebound to different type of values. For example:

```
phone = 5551234     # bound to a numeric value
phone = '555-1234' # now rebound to a string value
```

Therefore, it is impossible for the Python interpreter to know in advance what the actual type of a name is. When a message is sent to an object (usually through a name), the Python interpreter must first find out the following:

1. the object that is bound to the name;
2. whether the object has a method (implementation) for the given message or not; and
3. if yes, let the object respond to the message (execute the appropriate method).

It is easy to notice that the above is exactly how virtual methods work in other languages. In other words, all methods in Python are virtual methods.

We note that Python 3.0 provides a way to allow the methods to check for the types of objects through function annotation [1]. However, the process described above is still the same; and all Python methods are still virtual methods.

3.3 Inheritance

Inheritance is an essential feature of every object-oriented programming language. Without inheritance, a programming language can only be called at most to be object-based, but not object-oriented. However, Python's support for inheritance is a bit non-standard. In most other object-oriented languages, inheritance allows subclasses to inherit all data/behaviors from the parent class(es). The subclasses can then augment, modify, or suppress the inherited data/behaviors. On the other hand, Python subclasses only inherit behaviors, while data are not automatically inherited [3]. Although this will not be a major problem in most cases, sometimes it can introduce subtle bugs.

A Python subclass inherits from its parent class(es) all behaviors (methods implementation). The subclass can then be free to augment, modify and/or suppress these behaviors. Data in Python classes are mostly defined and initialized in the constructor (the `__init__` method). However, unlike other object-oriented programming languages (e.g. Java), which allow the constructor to implicitly invoke the constructor of the superclass, the constructor of a Python subclass will NOT automatically invoke the constructor of the parent class. There is in fact no mechanism to enforce that the constructor of the parent class is invoked at all. If the programmer "forgets" to call the parent class's constructor, then all definitions and initialization of data in the parent class are lost (since data are not inherited automatically), and later, when a method is trying to access these data, errors arise.

Therefore, it is very important to advise the users to remember to invoke the constructor(s) of the parent class(es) in the constructor of the subclass.

We now have looked at the three essential requirements for object-oriented languages and seen that Python more or less supports all these requirements (with some abnormalities). We will next look at some additional "common" features that make object-oriented programming a pleasure and investigate how they can be supported or simulated in Python.

4 Common Non-Object-Oriented Features

4.1 Abstract Methods and Abstract Classes

Abstract methods are those that the programmers only declare the signatures of the methods, but do not provide implementation. This allows programmers to declare the availability of the (abstract) methods without actually implementing them. Abstract methods may be very useful in situations. For example, say we have a `Shape` class as a parent class, and the different specific shapes (e.g. `Circle`, `Polygon`, etc.) as subclasses. It may be useful to declare a `draw` method in the `Shape` class, so we know that this method is available to all subclasses and how to invoke it, but it really makes no sense to implement the `draw` method in the `Shape` class as each specific class will draw itself differently.

Abstract classes are those classes that contain abstract methods, or the programmers specifically want them to be abstract. Abstract classes cannot be instantiated as the implementation is not really completed yet. For example, we cannot (should not) instantiate a `Shape` object as there is no concrete `Shape` – we can have circles, polygons, and so on, but we do not have concrete shapes, so it makes no sense to instantiate a `Shape` object.

In C++, abstract methods are marked with the `virtual` keyword; and any class with virtual methods is abstract class, and hence, cannot be instantiated. In Java, abstract methods are marked with the `abstract` keyword; and any class with abstract methods is abstract class, and hence, cannot be instantiated. Furthermore, Java programmers can also mark a class to be abstract even when it contains no abstract method, hence again, making it impossible to be instantiated. This is sometimes useful. For example, although each shape will be drawn differently, there may be an initialization sequence that is common to all shapes that will want to draw themselves. In this case, we can put the initialization code in the `draw` method of the superclass, and let the subclasses to invoke this superclass method to initialize the drawing.

In Python, there are NO abstract methods, and there are NO abstract classes, at least not up to Python 2.5. Norvig [5] suggested using a non-existing keyword `abstract` to cause exception when the abstract method is invoked. This depends on the fact that no one is defining a variable or function named `abstract`. Furthermore, the exception will be raised only when the abstract method is invoked; there is still no mechanism to prevent an object of a class with abstract methods to be instantiated. One work around is to use the same trick in the constructor (`__init__`) of the abstract class, making the constructor "abstract". This, however, will also prevent the subclasses to invoke this abstract constructor to finish initialization. Therefore, it is not a complete or satisfactory solution.

However, the Python development team is listening. As a result, abstract classes and abstract methods are now supported starting in Python 2.6. To use this new feature, we must import `ABCMeta` class and `abstractmethod` from module `abc`. Then we can mark a method to be abstract using the @abstractmethod decorator. Whenever we instantiate a class with abstract methods, we will get a `TypeError` exception. The following example is taken from the Python Documentation [6]:

```
from abc import ABCMeta, abstractmethod

class Drawable():
    __metaclass__ = ABCMeta

    @abstractmethod
    def draw(self, x, y, scale=1.0):
        pass

    def draw_doubled(self, x, y):
        self.draw(x, y, scale=2.0)

class Square(Drawable):
    def draw(self, x, y, scale):
        ...
```

4.2 Named Constants

Many languages support the idea of named constants. In Fortran, we use the keyword "`PARAMETER`" to mark a name to be constant; in C/C++, the keyword is "`const`"; in Java the keyword is "`final`". In fact, Java extends this idea to methods and classes. In Java, the "`final`" keyword basically means "fixed" and "non-changeable". Therefore, a final variable is an unchangeable name, i.e., a constant; a final method is a method that cannot be overridden; and a final class is a class that cannot be extended (subclassed).

However, there is no notion of constants in Python. (Python has immutable objects, but this is a completely different concept.) Every attribute in a Python class is changeable; every method in a Python class can be overridden; and every Python class can be subclassed.

In an Internet forum [9], there is a very interesting solution to the lack of final variables in Python. The idea is to override the __setattr__ method, which is implicitly invoked when a value is assigned to an attribute, to raise an exception whenever a new value is assigned to an existing attribute:

```
class WriteOnceReadWhenever:
   def __setattr__ (self, attr, value):
      if hasattr (self, attr):
         raise Exception ("Attempting to alter read-only value")

      self.__dict__[attr] = value
```

This actually works, to some extent. The problem is that ALL attributes in this class are now "write once only". This can be fixed by introducing some naming convention so the __setattr__ can check if a name should be a constant or not. For example, we can say that all variables with name beginning with "constant_" should be constants. Then we can modify the code as follows:

```
class WriteOnceReadWhenever:
   def __setattr__ (self, attr, value):
      if attr[:9] == 'constant_' and hasattr (self, attr):
         raise Exception ("Attempting to alter read-only value")

      self.__dict__[attr] = value
```

We note that it is still possible to directly modify self.__dict__ to bypass the checking.

5 Conclusion

We have investigated the object-oriented features of the Python programming language. Python supports encapsulation with limited information hiding; it has full support in polymorphism and inheritance. Therefore, Python fits the definition of object-oriented programming languages. On the other hand, there are many "add-on" features that are commonly available in many main-stream object-oriented languages but are missing in Python. We suspect that this is the main reason why Python is still not popularly used in the industries. However, this does not withhold Python to be an effective and ideal academic language to teach object-oriented programming. In fact, more and more colleges now use Python as their CS0/CS1 language. To efficiently use Python as an object-oriented programming language, we have to largely depend on advisement and documentation. We call this style of programming "Cooperative Object-Oriented Programming" and Python is one of the languages that depend on this style.

6 Acknowledgement

The publication of this paper is partially supported by Fayetteville State University's UCAMCS project which is funded by NSF grant NSF DUE 0631065.

7 References

[1] Alchin, "Pro Python, Advanced Coding Techniques and Tools," Apress, 2010.

[2] Damm, Hansen, and Thomsen, "Tool Support for Cooperative Object-Oriented Design: Gesture Based Modeling on an Electronic Whiteboard," in Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Hague, Netherlands, P518-525, 2000.

[3] Goldwasser and Letscher, "Object-Oriented Programming in Python," Prentice Hall, 2008.

[4] Nascimento and Dollimore, "A Model for Cooperative Object-Oriented Programming," Software Engineering Journal, Volume 8 Issue 1, P41-48, 1992.

[5] Norvig, "The Python IAQ: Infrequently Answered Questions," available at http://norvig.com/python-iaq.html.

[6] Python Documentation Team, "Python Documentation," available at http://www.python.org/doc/.

[7] Rossum, "Python Tutorial," available at http://www.python.org/.

[8] Silberztein, "NOOJ: A Cooperative Object Oriented Architecture for NLP," in Proceedings of the 5th INTEX workshop, Marseille, France, 2002, available at http://www.nooj4nlp.net/NooJArchitecture.pdf.

[9] Stack Overflow Forum, "'final' Keyword Equivalent for Variables in Python?" available at http://stackoverflow.com/questions/802578.

[10] Tucker and Noonan, "Programming Languages: Principles and Paradigms," 2nd edition, McGraw Hill Higher Education, 2007.

Cloud Computing and Security Attributes of Cloud Architecture

Atif Farid Mohammad, Hassan Reza
Department of Computer Science, University of North Dakota, Grand Forks, ND USA
atif.mohammad@und.edu, reza@cs.und.edu

Abstract

The advent of Cloud computing has introduced a way to add capabilities dynamically by increasing the capacity in new infrastructure using service oriented architecture without much of newer investments. These investments can include the issuing of new licensing, the training of users and the provisioning of external user friendly interfaces. Cloud also brought new extensions in the world of social and corporate sectors with available resources of available Information Technology. In the last decade, we have seen a tremendous growth in cloud computing in the IT industry. This paper provides an introduction of a security model, which does not negotiate with the required functionalities and capabilities in traditionally available web security models. This new model is called CSM 2.0 or Cloud Security Model 2.0, which is based on Service Oriented Architecture or SOA 3.0 and targets at improving security features over a traditional existing models, while not taking any risks by threatening other important features of the current security models. This paper also presents a survey of the different security risks that pose a threat to the cloud.

Key Words: Service, SOA, Software Engineering, System, Cloud Computing, QOS, Data Centre, Data Communications

1. Introduction

Cloud refers to the stipulation of different services required by users on-demand in the form of computational resources as a combination of services. The services are designed and developed using service oriented architecture or SOA. Any software application that requires frequent modification needs to be separated from the servicing applications, which is consistent and rarely needs to be updated. Service Oriented Architecture (SOA) is the application of this understanding on the knowledge management of a business.

Let us take an example to explore SOA. Each puzzle piece given in Figure 1.1 is a service provided by a retailer, travel agency, bank or a government service provider. These services are available around the world, as an offshoot of global business. Figure 1.2 depicts the servers providing these distributed services across the globe. Using the Cloud, these services can be accessed almost anywhere in the world. The magic of SOA works for consumers as well as industrial internal-operational and managerial users. The use of SOA generates the structure of these services shown in Figure 1.3.

Figure 1.1: Services

Figure 1.2: A distributed view of services [1]

SOA evolved in stages over the last few decades, since industrial automation increased. The services we use today process requests as input and produce output for customers, other systems or services. These systems or services orchestrate the data when generating messages among each other and to us as the user. The operational users of these services can monitor or manage many requests simultaneously. These operations can also be performed by a mediator service designed to follow the agreed policies and procedures among each of these services. Each service is owned and governed by a business entity and works within a certain body of rules, defined by the policymakers.

Figure 1.3: The magic of Service-Oriented Architecture

SOA provides a service data abstraction. This abstraction can be understood as a services messaging metadata, which can be in the shape of XML, XSD or other set industry standards providing interoperability to a user's request. The messaging between services is encapsulated for information-hiding purposes. A SOA solution for a bank as shown in Figure 1.4 is designed to maintain customer service by providing customer care as well as web self-service options for consumers.

SOA eases the development of ever-changing applications that compare data with stationary applications

while maintaining a decoupled relationship between these applications on a simultaneous basis as well as maintaining quality of service (QoS). SOA brought in a fresh approach for business information technology departments, making it easy to assemble and configure IT components like building blocks that can be combined to provide easy and fast solutions creation. For example, a bank provides a line of credit by checking a consumer's credit; an automotive parts seller checks the inventory; a shipping company maintains the shipping status for delivery to consumers, etc.

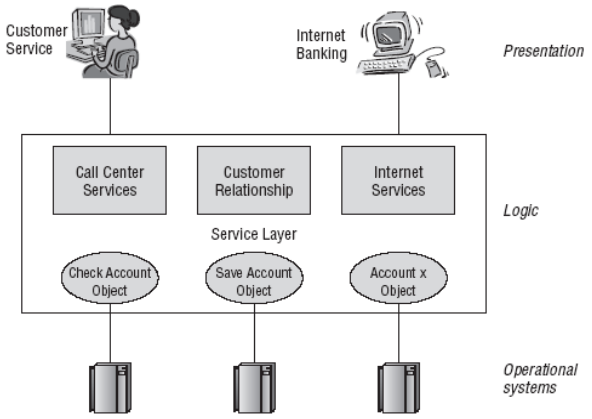


Figure 1.4: Bank [2] customer service solution.

SOA provides the framework to work modularly in the Cloud. Due to this flexibility, any business can adapt SOA and assemble its services as needed. A bank's services illustrate this in Figure 1.4. After changing a consumer base or adding new products or services for current customers or to woo new customers, these functionality components can be reassembled. This reconfiguration of service components is invaluable for saving money as well as time. New components can also be added to provide better or more services on the base functionality components. In the case of a merger or cooperative partnership with other related businesses, the components can be included as an additional service to the consumers at one platform. SOA provides the flexibility for businesses to improve and change with the pace of time.

As the world has become a global village using latest Cloud computing, there is a significant need for businesses to fulfill consumer needs by providing services globally. This pursuit presents some challenges as given below:

- Flexibility is required.
- Increasing demand of standardized services with seamless experience for consumers.
- Reduction in operational cost of these services by getting satisfactory results due to improved efficiency, which means users control their business, rather than technology.
- Services are mostly distributed.
- Getting regulatory approvals.
- Getting rigid information systems wrappers developed to get the services combined at one framework.
- Efficient use of existing resources.
- Services are heterogeneous.
- The patterns of services interaction are unpredictable
- The service(s) "front end" is less useful for testing purposes due to decoupled implementation on the backend.

SOA characterizes and provides the composite framework for the Cloud services based on open standards. Multiple services can be combined to serve a consumer at one independent Cloud; for example, a travel agent can provide car rental, hotel reservations or can book an entire vacation. This is a convenient way for the consumer to get several services with one phone call or one website utilization. Expedia [6] can be taken here as an example.

The services designed with transparency to the background-used technology and business processing management rules, for all internal and external users, offer the highest ongoing returns in terms of use and feedback for the industry's investment of skilled personnel's time. The utilization of open source technologies is recommended. Open source tools segregate service application front-end and back-end programmers from the specifics of any single platform and operating systems to produce transparent, portable, coded pieces of an application, which can later be combined as needed.

2. Background

Cloud has services serving users, and services are based on SOA, which is an adaptable and flexible approach—not a technology. SOA facilitates the utilization of reusable IT components to create new solutions over an existing framework of components. SOA provides platform-independent coupling of service components. This coupling can involve diversely developed service components in various languages, which can be maintained on several operating systems. The logical or functional separation of service components is provided by SOA. This separation allows software designers and developers to modify, test or redevelop and run these components on different servers before initiating them into a new lineup.

The challenges of the present financial climate can be resolved by software engineers and decision makers, such as CTOs or MIS managers, by aligning business needs, using service components to improve service to consumers. SOA is a boon to an enterprise looking to create service components in an agile fashion and reuse an existing infrastructure.

According to Schreiner and Lamb [3], systems of the future will be based on the concepts of SOA. Service applications will be composed of a number of individual

services running in the Cloud. As illustrated by Erl [4], service component application logic can be divided into two levels: a service interface, where loosely coupled services are available with their implementation and technology platform; and a service-using application level in which service application logic is developed and deployed on different technology platforms. These services communicate via open protocols.

The purchase planning scenario of a commercial or residential property is an example of a system where the property-selling agency can be provided with a flexible approach using SOA in the Cloud. The levels discussed above can be adapted as application and service interface levels. We now take a closer look at how SOA helps to plan a purchase, instantiate, and adapt the composed property purchase service. Initially, three services can be identified as:

i) "Property for sale" search service
ii) Offer submission to initiate a purchase service
iii) Credit check and mortgage service

In this example, partner service implementations are selected as follows. The "property for sale" service can be a local server provided by the real estate agency. The offer submission service can be a web service provided by another agency, or it can be assumed by law firm service providers. The credit check service is a web service provided by the financial department of a bank or a mortgage provider. Considering the strict security requirements of financial transactions, these services typically have fixed functional and non-functional requirements—i.e., security policies—that cannot be altered.

With SOA, these services can be combined software as a service or SaaS at one front-end platform on a website to provide the "property for sale" information or other services as mentioned above. To make this clear, we can take the example of Amazon's [5] store-front. Customers use a browser to get the displays on Amazon.com. The front-end website infers the customer's intent and triggers the services that do things like acquiring the data for the current on-sale products, or getting the customer's order. The important thing to note here is that the servicing components do not make proposals to the customers, and these services have no idea who they are talking to. These services are serving customers by getting data from some other services, which might be residing at some other server and can provide only product details. Some other services might be obtaining customer order details to invoke other services for shipping the products.

3. Related Work on Cloud Security Issues

As per Schneier, et. al, "Security is not a product - it's a process.". [7] Service security in the Cloud is not only a quality attribute, it is also regulated by governmental laws. There are several laws that can be identified to understand the need for Cloud security from the perspective of service providers, and from the perspective of the users. Some of the computer related crimes that are addressed by Criminal Laws [8] are:

- Unauthorized access
- Exceed authorized access
- Intellectual property theft or misuse of information
- Child pornography
- Theft of services
- Forgery
- Property theft (i.e., computer hardware, chips, etc.)
- Invasion of privacy
- Denial of service
- Computer fraud
- Viruses
- Sabotage (data alteration or malicious destruction)
- Extortion
- Embezzlement
- Espionage
- Terrorism

Cloud computing needs security of the cloud tested at an extensive level. To test any of the cloud service we need to know the base architecture of the Cloud security service(s). The base knowledge of Software architecture is a vital part to understand, that is the main core of software engineering. The Security of Cloud is an essential attribute, and is critical in making sure that there is no unauthorized access is allowed of any kind to the Cloud using by a corporation or even an individual. The Cloud service providers must consider, security of the service that they are about to provide to a consumer in the design process on earlier basis in system requirements engineering phase. This needs to be taken as a compulsory initiative at every architectural level in the service design, using SOA.

Service-Oriented Architectures (SOA) presents an advanced architectural concept with significance. Dorner et al. [9] have brought forward a few considerations of SOA in terms of End User Development (EUD). They analyzed the development of adaptable systems as a potential for SOA, proposing challenges that need to be solved to get an effective EUD. The authors' analysis is based on requirements for EUD systems and empirical studies, taken from earlier research work [10]. Dorner et al. have suggested in their study, that SOAs can be extended with structures for in-use modifications; the design of user-adaptable next-generation systems is also possible. EUD can also be suited develop Cloud service for the purpose of securing end-user as well.

With SOA-provided flexibility, the new tailorable systems can be produced, and platform independence can also be achieved. Services designed using SOA are formulated software applications, and this formulation is closer to business domains. Research has also indicated that the call for additional metadata of service descriptions is growing quickly, and the amount of data collected from experiences with a service needs to be stored for analysis of service and its future use. This data handling in terms of storage locations and synchronization raises issues and serious concerns about service performance. The service can have performance issues in terms of message communication to and from the user to the service provider due to this additional contextual information. Research has found that requirements of EUD of a service may involve extending protocol and server structures of SOA standards.

Cloud computing embeds almost all known computing devices as well as several of software, such as SaaS (Software as a Service) [1], PaaS (Platform as a Service) and several Operating Systems. It also utilizes as many data communication networks, such as local area networks (LANs), metropolitan area networks (MANs) and wide area networks (WANs). A recent survey by Cloud Security Alliance (CSA) & IEEE indicates the eagerness of corporate sectors to adopt cloud computing, major bottle neck is the security is needed both to hasten cloud adoption, while making sure to achieve regulatory drivers coverage for their daily activities. It is vital for organizations to look critically at security models to examine the confidentiality issues for their business critical tactless applications.

Due to several such gaps, it is yet not possible to provide guarantees that corporate data in the "cloud" is secured, if not impossible, as they provide different services like SaaS, PaaS, and IaaS. Each service has its own security issues [11].

In SaaS, the client has to depend on the provider for proper security measures. The provider must do the work to keep multiple users' from seeing each other's data. So it becomes difficult to the user to ensure that right security measures are in place and also difficult to get assurance that the application will be available when needed [12].

Next section details relationship of Service Oriented Architecture SOA 3.0 methodology relationship with software engineering. This discussion provides a base for an understanding of the Cloud Security Model 2.0 novel design.

4. SOA 3.0 and Software Engineering

Although software engineering has progressed in the past few decades, there are two realities still dominating the lives of real-world software engineers [13], whether it is traditional software applications development or a SOA service development, these two realities are:

A. The maintenance phase still costs typically 40% to 80% to an organization.
B. The existing legacy systems need to be maintained, and this maintenance involves the need to understand the existing legacy system.

There are several software applications used with the Cloud to serve as consumer-oriented services. These Clouds can be characterized by their intended services, such as business, engineering or scientific services. There are several factors involved in designing, developing, and deploying service software, otherwise known as "SaaS". These factors are critical to obtaining accurate solution(s). Software engineering can be divided into five significant aspects:

- **Ideas**
- **Concepts**
- **Structures**
- **Architectures** and **Operations**

Processes (functions) are the interrelated activities, operations and transformations by formulating, inputting or outputting that cause, create or contribute to performance of the software to meet the user's goals. As per IEEE 1471-2000: *Software architecture is the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.* Software is developed using the following key components:

- Conceptual modeling
- Well-defined processes
- Power tools (Computer language(s))

Software engineers perform several types of tests to extract data; which is then processed. Suitable or non-suitable results are recorded during and/or after the experiments. This type of data processing, called conceptual modeling [14], [15] is performed by software QA testers. "*In a few systems, the conceptual view plays a primary role. The module, execution and code views are defined indirectly via rules or conventions, and the conceptual view is used to specify the software architecture of a system, perhaps with some attributes to guide the mapping of other views* [16]."

Software development is not an easy task. It is important that the software engineer use a mix of software engineering practices to prepare the software for delivery to the stakeholders with an eye to upcoming changes. Operational users must be well-informed at each step of software development to achieve near to accurate product. One important aspect in a software development cycle is its time of development after an acceptable design is agreed upon by all stakeholders. During this timeframe, the changing needs of business also have impact, which can increase development time or cause a failure of the software

at deployment. Such problems can increase cost and make the whole process effort-intensive.

Software as a Service or SaaS in relation to Cloud Computing needs to be designed and developed in a given time within available resources. This type of development is entirely different, as some students design and develop a certain application as their term project, which they can code as a single program and test for accuracy. SaaS usually contains several applications woven together to resolve several challenges faced by an enterprise for its private Cloud. These applications are compatible services for a certain organization. Any successful software as a service or SaaS is usable, cost-effective, and maintainable as well as evolvable. SaaS has a base architecture, and next section of this paper details a proposed novel design of Cloud Security Model as CSM 2.0 to be further developed and tested as a future work for upcoming research in Cloud Computing at Soft-End Lab at University of North Dakota.

5. Proposed Cloud Security Model

It is vital that the use of cloud is to be secured to prevent any security breaches for both users as well as service providers [17]. In this research paper, we present a Cloud Security Architecture using Cloud Security Model (CSM 2.0) to make sure that both data and user requiring this data is protected under the federal law of ITAR/EAR [18][19]. This model is based on Layered Architecture and looks into a Cloud domain through the Resource Management as shown in the given figure.

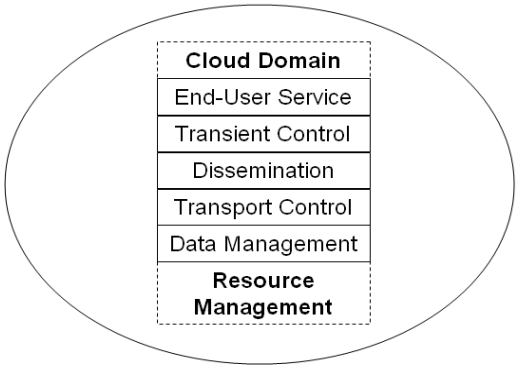


Figure 5.1: Cloud Security Architecture [17]

The proposed architecture can be understood by the following Figure 5.3. CSM 1.0 contains a consumer login service, which needs to adhere the procedures and protocols provided by the service/service provider. Every service has an end-point; we call it Edge, which is to bind the consumer to use the service on the basis of contract accepted by both consumer and service provider. Communication among user or consumer is the key transient control between both for transparent use of the service(s). These procedures and processes are depicted in Figure 5.2.

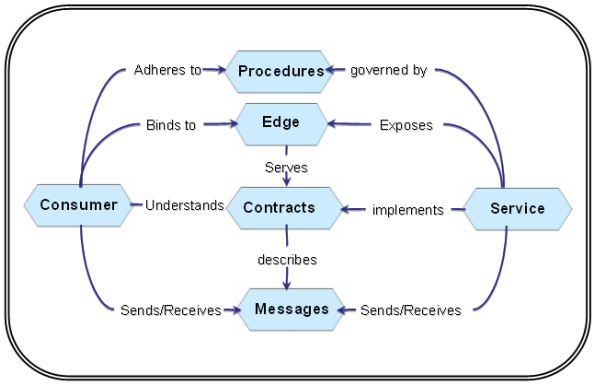


Figure 5.2: Cloud Security Model – CSM 1.0 [17]

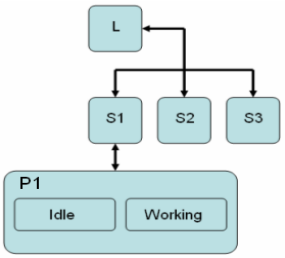


Figure 5.3: A functional example of an SRD

The processing operator **P1** connects and relates elements to show the logical flow to construct a block in order to illustrate the performance of a desired system. The performance of a system is illustrated in two foremost ways, expressed by the combination of communicational links and conditional operators. A processing operator can have several inner processing operators related to each other directly or indirectly; one or several of these processing operators can be used to serve some other block of the system on a simultaneous basis. Let us assume the notations as shown below:

- L = List of Services
- S1 = Service 1
- P1 = Process Operator 1
- P1.1 = Idle Service
- P1.2 = Working

L is a **list of services** available in the Cloud containing a combination of **S1**, **S2** and **S3** (the **services** provided by a service provider) and **D1** (a service provided by a data centre in the Cloud). There can be two major processes a service can be in—0 and 1. The service is **Idle** or **Working**. The use of CSM 1.0 protocols provides us a secured way to use any further services from this point onwards.

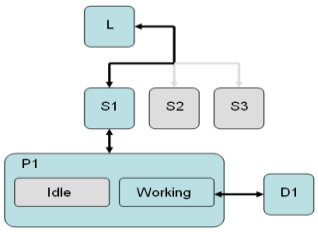


Figure 5.4: A working service prototype

Let us consider the service S1 is a Security Check Point 1 and the user's authentication is clarified for further processing. The service might be idle due to no job being needed to be done at this moment in time. The working service might also be idle, as there can be a delay in receiving some messages for next level of authentication combining user's as well as service provider's authentication for other services or a data centre of some kind. P1.1 and P1.2 can further be drilled down. The Figure 5.4 shows a prototype of getting required information for the service user of **L** from data centre service **D**.

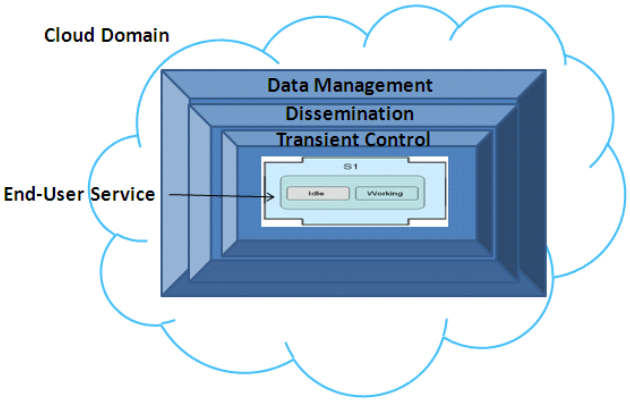


Figure 5.5: Three layered Cloud Security Model 2.0

The service **S1** as shown in Figure 5.5 is the actual service built as an application block. This service block S1 connects with another service, **D1,** which is a service provided by a data centre in the Cloud and is assigned with first encrypted key **K1**, upon third level of authentication key **K3,** which is to be assigned by Dissemination level as the main locked layer key **K2** to issue clearance to get the required data for the user of service **L**. The processing block as shown in earlier Figure 4.4 can be seen as an independent processing operator connecting two services by communication of the requestor's requirement, and the operator gets the resultant data set(s) and delivers it to the requestor.

Let us look into further details of these services; **L** is a listing service of several services. For simplicity's sake, we will take the service **L** as given below:

- Detailed display of services available by a provider has applied CSM 1.0 protocols.
- Facilitates user's request as input to get a resultant data set at each entry and exit point of the service.

The service **S1** provides the following features:

- Generates a query on the request input once cleared from CSM 1.0 protocols.
- Stores the data for future feedback for service designer
- Manages bandwidth rates of data set receiving
- Delivers the data set(s) to service **L** upon clearing from CSM 1.0 protocols.

The service **D1** provides the following features:

- Receives query from **S1** cleared by CSM 1.0 protocols.
- Processes the resultant data set(s)
- Delivers to **S1** upon clearance from CSM 1.0 protocols.
- Stores the query for future use for some other user.
- Stores query snapshot for feedback for service designer.
- Self-manages the storage usage.
- Manages bandwidth rates of every query.
- K1 contains detail of data center originating IP address, as well as physical address with service provider's name and company license info.
- K3 contains detail of requestor/user's originating IP address.
- The main process of K2 is to confirm that both user and data provider are with ITAR/EAR defined boundaries;

In SRD, a database is also considered a service. Here are a few real-life examples given below to understand the use of database services to the users on an everyday basis:

- An owner/operator of a transport company can use the service **S2** using CSM 2.0 protocols to enhance his business by maintaining and adding the data associated with his customer organizations. This database service will keep a record of load pick-up and delivery dates for all the customers (organizations) provided by the transport company. If this transport company is planning a special pre-summer promotion, the database service will be able to generate results of the customers to target for advertising to increase business.
- A chief organizer of ABC party can take advantage of the same database service **S2** using CSM 2.0 protocols for a fundraising event. The mailing list generated by the database service is made up of a wide range of organizations with contact person's data of every organization arranged as per the area or city, town or a state. These people can be invited with reference to the transport company's owner for a fundraising dinner for the ABC party for election.

In the case of a merger between the manufacturer of a certain product, such as a printing company, and the transport company, the service S2 using CSM 2.0 protocols

can be used to inform customer organizations of the newly available services of printing for clients to go with the transportation services as well.

The use of CSM 2.0 will prevent unauthorized access of any data hosted within US borders critical to the ITAR/EAR munitions list to any of the users of both merged concerns as given in the above example to access restricted documentation from outside of the country, except been exempted to get access to under the given provisions of the law.

6. Future Work

It is critically significant that a tradeoff analysis needs to be done by a Cloud service designer in order to clearly understand the impact of security on the overall Cloud services architecture. This analysis will direct the selection of an optimal architecture. SOA 3.0 can be considered as most advantageous architecture which will satisfy the security requirements, while supporting reasonable tradeoffs by reducing the negative impacts of security vulnerabilities on the other desired quality attributes such as performance and usability of the services provided by the providers to the Cloud users.

In the next step, we will focus on optimizing the proposed Cloud Security Model for dealing with complex structured Services and strengthening security of cloud computing model by implementing CSM 2.0 on real-time services in use of our daily lives by initiating a test Cloud with in UND Computer Science department. We also will be looking at details related to ITAR/EAR, (which have United States munitions list) to make sure that compliance issues in relation to proposed CSM 2.0 are properly covered.

7. Conclusion

As described in the paper, though there are tremendous advantages in using cloud-based systems, there are yet many realistic problems which have to be solved. Cloud is a union of several hardware and software platforms. There are several hybrid technologies being utilized to provide the services in the cloud by many service providers. This paper sheds light on few of the several issues dealing with Cloud and specifically the security issues and research work done to find suited solutions to resolve security problems. It also encompasses a proposal of Cloud Security Architecture based on Layered Architecture approach. This approach is presented as Cloud Security Model and is named CSM 2.0, with expansion of three services with an addition of encrypted internal keys to find the source of request and destination, which needs more future enhancements and modifications with the incorporation of systems functional testing. CSM 2.0 has potential to be enhanced to achieve a favorable methodology to resolve the issues related to Cloud Security, reliable availability of the desired results from the effective use of Cloud's available resources.

References

[1] A. F. Mohammad, E. S. Grant; Cloud Computing, SaaS and SOA 3.0: A New Frontier. Cloud Computing and Virtualization 2010 International Conference, Singapore May 2010

[2] M. Rosen, B. Lublinsky, K.T. Smith, M. J. Balcer. Pulished. Applied SOA: Service-Oriented Architecture Design and Strategies. Wiley Publishing Ltd. 2008.

[3] R. Schreiner, U. Lang; Protection of complex distributed systems. Proceedings of the 2008 workshop on Middleware security. Pages 7-12. 2008.

[4] T. Erl. Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR, 2005.

[5] Amazon: http://www.amazon.ca/; Accessed on April 03, 2011.

[6] Expedia: http://www.expedia.com; Accessed on April 10, 2011.

[7] Schneier Bruce, Secure System Engineering Methodology, http://www.counterpane.com/; Accessed on April 15, 2011

[8] H. P. Tipton and M. Krause, Information Security Management 4th edition, Auerbach, United States of America, 2000

[9] C. Dörner, V. Pipek, M. Weber, V. Wulf. End-user development: new challenges for service oriented architectures. in Proceedings of the 4th international workshop on End-user software engineering, pp. 71-75, May 2008

[10] F.P.J. Brooks. No silver bullet: essence and accidents of software engineering, IEEE Press, pp.10-19, 1987

[11] B. R. Kandukuri, V.R. Paturi, A. Rakshit. Cloud security issues.In: IEEE international conference on services computing, p.517–20. 2009

[12] V. Choudhary. Software as a service: implications for investment in software development. In: International conference on system sciences,p.209. 2007

[13] Glass, L. Robert. Facts and Fallacies of Software Engineering. Addison Wesley, 2006

[14] H. Liu, and D. Gluch. Conceptual modeling with the object-process methodology in software architecture. J. of Computing in Small Colleges, 19 (3), 10-21. 2004

[15] D. Dori, M. Choder.Conceptual Modeling in Systems Biology Fosters Empirical Findings: The mRNA Lifecycle. PLoS ONE 2(9): e872. 2007

[16] C. Hofmeister, R. Nord and D. Soni, Applied Software Architecture, Addison-Wesley, 2000

[17] A. F. Mohammad, R. Marsh, E. S. Grant. Cloud Security Model using SOA 3.0 - CSM 1.0: A New Frontier. 2nd International conference on Cloud Computing and Virtualization 2011, Malaysia, April 2011

[18] ITAR: http://www.pmddtc.state.gov/regulations_laws/ itar_official.html; Accessed on April 19, 2011

[19] EAR: http://www.gpo.gov/bis/ear/ear_data.html; Accessed on April 19, 2011

Cloud Computing: Past, Current and Future

H. Mcheick[1], F. Obaid[2], and H. Safa[3]
[1]Department of Computer Science, University of Quebec at Chicoutimi, Chicoutimi, Quebec, Canada
[2]Department of Computer Science, Lebanese University, Beirut, Lebanon
[3]Department of Computer Science, American University of Beirut, Beirut, Lebanon

Abstract - *With the augmentation of data, the organizations need to provide better and faster secured data access to customers. Cloud computing came as a godfather of the new generation of data warehousing and data access through the network. This new way of deploying and using data solved many of data problems. However, as any new technologies, it has its own issues and needs. In this article we are going to discuss and survey briefly this subject to make an overview of what might be the past, current and future of cloud computing. This might be at the same time the future of information systems around the world and the future of many organizations.*

Keywords: Cloud Computing; Data Access; Data Deployment; distributed and localized Data.

1 Introduction

1.1 Issues

When it rains, in few minutes we can see the streets full of water. What gives clouds the ability to do such a seen in such a time can be easily expressed by the following comparison: The difference between a raining cloud and a hosepipe is that a cloud drops water in parallel. From this, comes the idea of cloud computing (CC), we don't have to have many servers to make parallel transactions [1]. We can make them from the same site, which will ease the update time and do us big favour in many ways that we'll see in this article.

Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [13].

Cloud computing is the most popular notion in IT today; even an academic report from UC Berkeley says "Cloud Computing is likely to have the same impact on software that foundries have had on the hardware industry." They go on to recommend that "developers would be wise to design their next generation of systems to be deployed into Cloud Computing". While many of the predictions may be cloud hype, some believe the new IT procurement model offered by cloud computing is here to stay [14]. Whether adoption becomes as prevalent and deep as some forecast will depend largely on overcoming fears of the cloud.

CC represents a new way in how to use and think about computing resources and services. Hence, clouds represent a new way in which users store, access, and utilize data and high-powered resources (IT infrastructure, software and services) remotely via Internet without the need to build large on-site IT systems. The "cloud" is a network of data centers providing commoditized computing power to end users portioned in an as-needed basis to clients in a fashion similar to electricity or water [2].

This gives individuals and small companies the opportunity to access vast computing resources with high-speed, high-quality, and non-discriminatory networks they couldn't use before. Nevertheless, the complexity of this new technology puts pressure on the existing system: First, the use of distributed systems would change to the use of clouds, so all distributed systems need to be changed and companies that make these systems would lose customers and money, unless these companies begin making cloud systems. Second, the computers, networks, and many other components (software and hardware) need to be developed to meet the needs of cloud systems.

Thus CC represents a disruptive innovation. It is a new alternative to acquiring software, platforms, and IT infrastructure as your business grows. Services are now accessed over the web on demand from any location via a broadband framework [2]. The CC technology has many challenges such as failures detection, scalability, security, specific legislations and licensing models, etc. For example, the new threats require new constructions to maintain and improve security. These challenges will be discussed later in the article.

1.2 Main Aspects of Cloud Computing

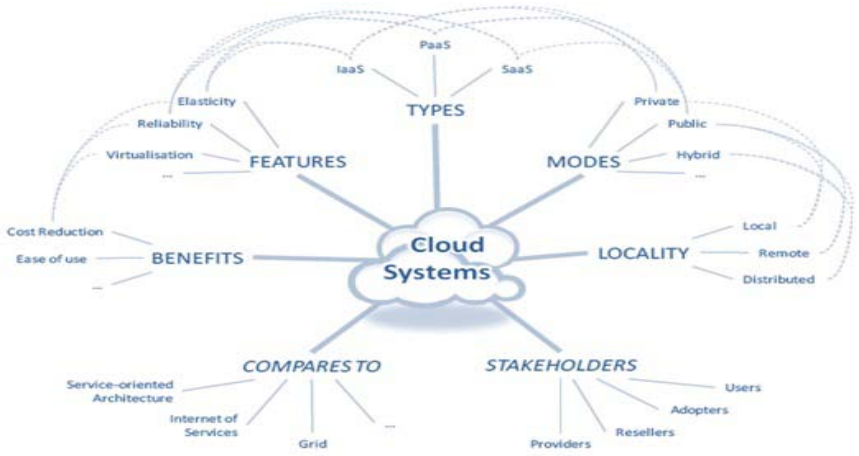


Figure-1: Main aspects forming a cloud computing system [3]

The main aspects of CC are:

- Types [3]:
  - Infrastructure-as-a service (IaaS): it replaces a company's entire server room and network through virtualization technology in order to cut costs and improve flexibility. Examples: Amazon Web Services, Rackspace, Savvis, HP, IBM, Sun and Google Base.
  - Platform-as-a-Service (PaaS): Delivery of a computing platform and software stack over the Internet to develop and deploy applications without the necessity to manage their own hardware and software layers on-site. For example, Amazon Elastic Cloud [15], EMC Atmos [16], Aptana [17] and GoGrid [18] are providing these services preventing users the mammoth costs of buying hardware, software and related technology as well as, to maintain and supporting their IT infrastructures.
  - Software-as-a-Service (SaaS): Provide online email applications, free services, limitless storage, and remote access from any computer or device with an Internet connection. Example: online email (Google, Yahoo and Microsoft).
- Features [3]: Elasticity, Reliability, and Virtualization
- Modes [3]:
  - Public cloud: Companies migrate their infrastructure to an existing public cloud.
  - Private cloud: Companies build their own clouds or participate in an interagency cloud.
  - Community cloud. The cloud infrastructure is shared by several organizations and supports a specific community that has shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be managed by the organizations or a third party and may exist on premise or off premise.
  - Hybrid cloud: Companies build their own clouds to handle its main applications and use public clouds to support low-priority applications.
- Benefits: i) Cost reduction: Adapt to changing consumer behaviour and reduce cost for infrastructure maintenance and acquisition, and ii) Ease of use: Through hiding the complexity of the infrastructure.
- Locality: Local, remote, and distributed.
- Compared to: SOA, Internet of services, and Grid
  - Stakeholders:Users, Adopters, Resellers, and Providers.

2 Market Model of Cloud Computing

2.1 Competiveness of CC

Since nowadays computing power is cheaper, the increase in revenues, customer satisfaction and market-share is becoming obviously clear [3].

Its competitiveness is associated to non-functional capabilities that represent qualities and properties of a system,

economic considerations, and technological challenges that arise from realizing non-functional and economic aspects [3].

- Economic aspects [3]:
  - Reduce the cost of the cloud system that takes into consideration the scalability and pay per use (building cost according to resources consumption) aspects.
  - Reduce the time to market for small and medium enterprises.
  - Companies will spend money on operational expenditure instead of capital expenditure that is required to build its own infrastructure.
- Technological aspects [3]:
  - *Virtualization*: Virtual infrastructure.
  - *Multi-tenancy*: Implies potential issues.
  - Security, Privacy and Compliance: dealing with sensitive data and code.
  - *Data Management* for storage clouds, where data is distributed across multiple resources.
  - *Metering* of any kind of resources / services is a condition for the cloud elasticity.
  - *Tools* to support development, adaptation, and usage of cloud services.
- Non-Functional Capabilities [3]:
  - *Elasticity*: capability to adapt to changing, dynamic integration.
  - *Reliability*:
  - *Quality of Service*
  - *Agility and adaptability*: On-time reaction and adaption to changes.
  - *Availability*: The ability to assure redundancy for services and data.

Porter's Five Forces. The figure 2 shows five forces of CC. These forces are described as follows.

Figure-2: Porter's five forces (Market model) [4]

- New Market Entrants: Many immature entrants that have not had the time to acquire deep business expertise, is expected which will present consumers with a significant challenge in identifying suitable implementation partners [5].
- Suppliers: Suppliers are the main player in the CC market, they own and operate CC systems to deliver services (Software, Platform or Infrastructure) to consumers. [5]
- Buyers (Consumers): The consumers will set the standards in the industry and partly drive the aggregation of players through more demanding requirements. [5]
- Technology Development: Experts agree that the influence of regulators will be a crucial factor in the coming years.
- Cloud Market: The cloud system itself represented as a market [1].

2.2 Entrants and Networks

The first time CC was to be introduced to the world was through Amazon Elastic Compute Cloud (Amazon EC2). [6]

Amazon Web Services has since grown into a new business that encompasses compute and storage infrastructure services and new middleware services that Amazon EC2 customers can leverage, such as the Amazon Simple Queue Service. [6]

Many followed Amazon to this new IT outsourcing model, The early entrants to CC come from two camps:

Internet services companies. The first camp (led by Amazon Web Services, Salesforce.com, and Akamai) is turning its infrastructure management services into profit centers serving internal and external customers [6].

Hosting providers. The second camp is the forward-thinking hosting providers that see clouds as the next step in the evolution of their business models. This group is comprised of large multinationals like Terremark and smaller players, such as Layered Technologies and XCalibre that see clouds as new offerings that can differentiate their businesses. Startups like Enki, from NetSuite's data center pioneer Dave Durkee, are basing their business on this model [6].

Many hosting providers (and several enterprises) are building clouds using 3Tera AppLogic, a grid engine that has evolved into full-blown CC infrastructure software. This technology works across physical and virtualized servers and has built-in high availability, virtualized storage, and per-use reporting. Unlike most other grid offerings, AppLogic works with most any type of application without redesign or programming to a grid API [6].

3 Current and Future of CC

3.1 Current Status of Clouds, the S-Curve

Bandwidth, perception, loss of control, trust and feasibility were the challenges that confronted the presence of CC service in the past. Many of these challenges were overcome by our new technologies and others will be in the future. Which means that this service moved from virtual to real and will be advanced as our technologies advance. [7]

The desire to reduce costs and add flexibility to huge enterprises are the most effective reasons that will make CC commonly used. [8]

In the coming few years, we would expect more and more companies to adopt for the cloud solution. Many companies both big and small, are currently seriously considering shifting to cloud services because of the many benefits to a cloud solution. The current status of CC is past the Innovators stage and gently moving up the lower cusp of the S-curve into Early Adopter stage (Figure 3) [9].

The advantages and benefits of CC are categorized in three categories:

- Centralization: [7]
  - Competitive advantage in data access.
  - Huge flexibility in data access.
- Cost: Few huge clouds cost less than thousands of large local servers. Less materials, less areas, fewer employers.
- Environmental effects: [7] i) Less need for infrastructure, ii) Less need for hardware, iii) Huge reduction in energy consumption.

The advantages and benefits of CC are categorized in three categories:

- Centralization: [7]
  - Competitive advantage in data access.
  - Huge flexibility in data access.
- Cost:

  Few huge clouds cost less than thousands of large local servers. Less materials, less areas, fewer employers.
- Environmental effects: [7]
  - Less need for infrastructure.
  - Less need for hardware.
  - Huge reduction in energy consumption.

Hence, a few large data centers with clouds are likely to be more 'green' than millions of smaller but already large data centers. For these reasons, CC is sometimes referred to as a Green information technology as it significantly reduces the carbon footprint [10].

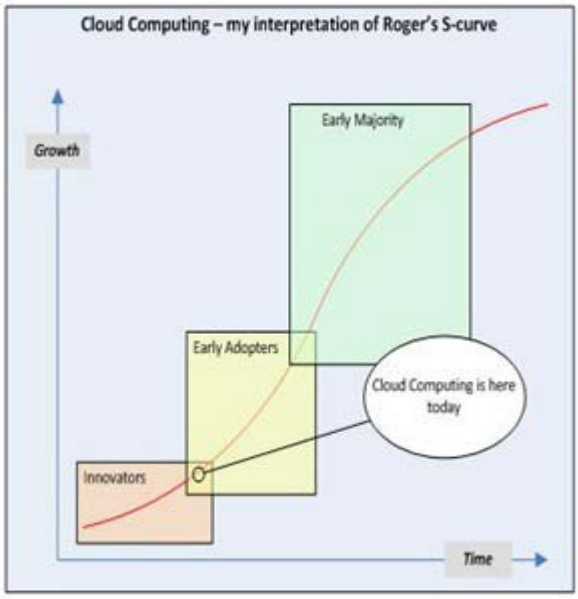


Figure-3: Current status of cloud computing on the technology maturity curve [7]

3.2 Future and R&D

In the future, more cloud adoption is inevitable. However, for a better assessment of the future of clouds, we will discuss some of the current challenges and gaps that are hindering the rapid evolution of this technology. Such gaps and challenges can be divided into two categories: technical and non-technical. After discussing the gaps, we will suggest corresponding set of suitable R&D steps that needs to be taken to overcome these challenge and make a better use of the CC system (Attention: some gaps might happen in the future and will need to be taken care of). [3]

Technical: [3]

1. The *ability to detect failures*, adapt to the required scale of resources., ensuring continuous availability of such resources, and meeting clients expectations in terms of quality.
2. *Privacy and Security*.
3. New security holes will appear with hackers advancing in their efforts.
4. *Adaptability*. Example: If a CPU is added to a virtual machine that is already in use, the running code should be able to adapt and make use of the additional resource without having to be restarted or even adapted.

Non-Technical: [3]

5. All data, resources, applications, and services need specific legislations and licensing models depending on the location they are hosted in. The challenge with such a step is the different laws put forward by different countries which might make it hard for hosting such a service in some locations.
6. Huge investment and Intensive research.

– Solutions: [3]

1. New segmentation concepts and distributed programming models.
2. Solution: Data should be simultaneously protected in a form that addresses legislative issues with respect to data location and be manageable by the system.
3. Solution: Imposing clear legislation models regarding jurisdiction over the hosted data and its distribution in other countries.
4. Solution: Developing programming models to provide adaptability.
5. Solution: All legislation issues should be properly addressed by respecting the uniqueness and distinctive nature of some countries and playing by their rules.
6. Solution: New expert systems and best use recommendations.

4 Discussion and Recommendations

Discussion. No matter what your applications and requirements are, cloud services could present the solution for such necessities. Companies might not fully switch to a cloud service but rather go for partial hybrid switch as it serves their specific needs. Microsoft for example, adopts such a strategy, and thus, they have built their cloud solutions based on hybrid (Figure 4) type offerings [11].

As an easy and simplified guideline for companies thinking of cloud as a potential solution, we recommend the following analysis as a potential startup [11].

When considering cloud, one should go into data physics to study the time and cost consumption of such technology. Data physics correlates between processing elements and the data on which they operate. The fact that CC stores all data in a cloud rather than physical hardware (hard disks) imposes the issue of time consumption taken by data to move from the cloud to the relevant server for processing. [11]

The governing equation for time consumed by data to move from its storage location to the processing server is:

*time = bytes * 8/bandwidth*

where the bandwidth to and from the cloud provider is the bottleneck. Therefore, the memory capacity serves as the key bottleneck for virtual machine density and improving such a key factor plays a major role in the decision of "cloud computing" deployment. [11]

So as a simple rule of thumb, if the time calculated to move data from cloud to server or cloud to cloud is more than the relative data processing time, then implementing such a strategy becomes worthless [11].

Figure-4: Hybrid clouds with private and public cloud models [11]

Recommendations. We now see CC as a shift from using many big servers to using one giant server, from distributed data (of the same company) to centralized, though, it's still to be distributed when 2 companies agree to share data. For the first company, its clouds are private and local, and the second company's data are public and distributed. For the second company, it's vice versa. For example, if Facebook and Twitter agree to share information about their users, Twitter data are public and distributed for the Facebook company, but these same data are private and localized in Twitter eyes. Using clouds is to simplify what was complex and ease the access of what was difficult to access, many other benefits of using clouds, some were mentioned in this article, some weren't, and some are to be discovered. [1]

We recommend the following:

1. Use CC systems whenever possible.
2. For the best use of these systems we recommend spending some time and money on doing researches because it's always better to lose a few thousands of dollars than to risk a few millions.
3. CC might be a benefit for some companies and a disaster for other companies, so when considering cloud computing we must do our own researches and not to totally depend on other's researches.
4. Companies should contribute together in the use of clouds by sharing "none personal" data, it has almost no disadvantages and could have many benefits.
5. CC systems could be more useful, all we need is to upgrade our technologies to meet the requirements for such a useful system.

6. Cloud fears largely stem from the perceived loss of control of sensitive data. Current control measures do not adequately address cloud computing's third-party data storage and processing needs. New visions propose to extend control measures from the enterprise into the cloud through the use of Trusted Computing and applied cryptographic techniques. These measures should alleviate much of today's fear of cloud computing, and, is believed, have the potential to provide demonstrable business intelligence advantages to cloud participation.

5 Conclusion

Cloud computing (CC) offers an exciting opportunity to build data structures that promise to solve problems associated with economic modeling, terrorism, healthcare and epidemics, etc… and to bring on-demand applications to customers in an environment of reduced risk and enhanced reliability [1].

Moreover, clouds could play a major role in climate change as they have proved to be a reliable green option that will contribute to reduction of carbon footprints. CC promises to reduce run time and response time of deploying applications, increase the pace of innovation, and lower entry costs, all while increasing business agility. As much as it seems promising, successful deployment of cloud technology requires change of design for current existing applications and cannot just be unleashed on the cloud as it is [1].

Design enhancements should focus on improving scalability factors to ensure service could keep up with demand requirements. If such relevant design upgrading is applied, deploying cloud technology will provide customers with a unique and genuine opportunity to reap the benefits of achieving the competitive advantage of a flexible, scalable, high speed, error reduced, and environmental friendly application solution.

6 References

[1] Writers own information, ideas, understanding to the subject and point of view."Cloud Computing". Available at: http://www.ccianet.org/CCIA/files/ccLibraryFiles/Filename/000000000151/Cloud_Computing.pdf

[2] "THE FUTURE OF CLOUD COMPUTING - OPPORTUNITIES FOR EUROPEAN CLOUD COMPUTING BEYOND 2010". Available at: http://cordis.europa.eu/fp7/ict/ssai/docs/cloud-report-final.pdf

[3] "Cloud Computing in Research - SYSTEMATIC THOUGHT LEADERSHIP FOR INNOVATIVE BUSINESS". Available at: http://www.ccice.org/files/Maik_Lindner.pdf

[4] "Cloud computing Forecasting change". Available at: https://www.deloitte.com/assets/Dcom-Global/Local%20Assets/Documents/TMT/cloud_-_market_overview_and_perspective.pdf

[5] "Is Cloud Computing Ready For The Enterprise?". Available at: http://www.3tera.com/download/Forrester-Cloud-computing-report080307.pdf

[6] " Cloud Computing; The Past, The Present, The Future (Part 1)". Available at: http://www.windowsecurity.com/articles/Cloud-Computing-Past-Present-Future-Part1.html

[7] "Hardware vs. Software: The Defining Technology Battle of This Decade". Available at: http://www.xconomy.com/boston/2010/01/20/hardware-vs-software/

[8] " Cloud Computing Climbs Adoption S-Curve". Available at: http://www.informationweek.com/news/software/info_management/228900680?queryText=

[9] "Cloud computing's green paradox". Available at: http://news.cnet.com/8301-19413_3-10428065-240.html

[10] "INTRODUCTION TO CLOUD COMPUTING ARCHITECTURE - White Paper - 1st Edition, June 2009". Available at: www.pcworld.com.vn/Handler/FileDowload.ashx?fileId=5151

[11] " Cloud computing leaders". Available at: http://www.cloudcomputingleaders.net/

[12] Peter Mell and Tim Grance. "Effectively and Securely Using the Cloud Computing Paradigm". NIST, Information Technology Laboratory, 2009.

[13] Armbrust,M., Fox,A.,Griffith,R. et al. Above the Cloud: A Brekely View of Clod Computing. UCB/EECS-2009-28, EECS Department, University of Califonia, Berkely,2009.

[14] http://aws.amazon.com; Accessed on February 21st, 2011

[15] http://www.emc.com/products/family/atmos.htm; Accessed on February 21st, 2011

[16] http://docs.aptana.com/docs/index.php/Aptana_Cloud Docs_Wiki; Accessed on February 21st, 2011

[17] http://www.gogrid.com; Accessed on February 21st, 2011.

Model Based Engineering of Ground Based Risk Mitigation System

Hassan Reza, Feifei Gu, and Mark Askelson
School of Aerospace Sciences
University of North Dakota 58202
USA

reza@aero.und.edu

Abstract

In this paper, we describe how AADLs can be extended by using Colored Petri Nets (CPNs) to simulate the behavior of an avionic system known as the Ground Based Risk Mitigation System (GBRMS). To this end, we provide a set of simple mapping rules that can be used to translate AADL representations of a system to CPN representations. The CPN representation of a system then has been used to simulate the dynamic behaviors of the GBRMS.

Keywords: *ADL, AADL, Color Petri nets, Model-Based Engineering, OSATE, Aerospace Software Engineering, Simulation, Behavioral Modeling.*

1. INTRODUCTION

Architectural Analysis and Description Languages (AADLs) and the supporting toolset known as OSATE (Open Source Architectural description Tool Environment) are model-based system engineering notations and environments that are becoming an essential part of developing highly complex and safety critical systems. The design of AADLs was motivated by the need to specify and analyze the various quality or dimensions of a safety critical system in an integrated environment in which both core hardware and software elements of a system can co-exist. This co-existence feature, in turn, will provide, among other things, a common ground that facilitates the communication among stakeholder (e.g., software, hardware, and control engineers) and increases productivity, reuse, and maintainability by promoting software engineering principles (e.g., abstraction, modularity and locality, and separation of concern).

AADL is based on many years of collective work and experiences with ADL (architectural description language) [1] [2] [9]. AADL was originally developed by the *Honeywell Corporation* under SAE (Society of Automotive Engineers). The underlining formal theory of AADL is based on MetaH [3] [6] [8] [9], which is a domain specific ADL used for specification and verification of RT systems.

In general, the AADL-SAE (see appendix A) standard has provided (at least) the following features:

1. Supports model representations (e.g., textual and graphical);
2. Provides an XML interchange format that allows model transformations, integrations, and/or navigations between different models (e.g., textual to graphical representation or vice versa);
3. Provides a UML [4] profile representing AADL as a specialized modeling notation within UML/MDD [13];
4. Supports integrateability with existing commercial and/or open source tool solutions and plug-ins written in [11][12];
5. Supports agreement with various programming language (e.g., Ada and C);

6. Supports an error model annex to model and analyze the reliability and safety of a system.

Using OSATE, an engineer is able to perform (at least) the following tasks: translate AADL to MetaH, syntactic consistency analysis, performance analysis, safety analysis, security analysis, and reliability analysis.

Although AADL and its tool support, OSTATE, support many of features that are required for simulating, prototyping, and analyzing the quality of a system at a high level of abstraction, AADL and OSTATE lack a very important capability needed to simulate and/or analyze the functional correctness of a system. This shortcoming of AADL-OSATE is attributed to the limitation of MetH (underlying formal theory of AADL). This deficiency of AADL may result in serious problems because the development of safety and mission critical systems demands the highest level of dependability.

On the other hand, Color Petri Nets (CPN) are a high class of Petri Nets having well-defined semantics with precise representation of places, transitions, arcs, and inscriptions (i.e., collective term to refer to the tokens, their types and values, constraints, and functions/expressions) [5]. CPN (see Appendix B) has a well-established body of theoretical work and results for specification and verification of static and dynamic behavior of highly complex and concurrent systems in various domains such as manufacturing, avionics, military, and medicine. Using CPN engineers should be able to apply CPN theory and supporting tools to prove the properties of the system under construction (e.g., the absence of livelock/deadlock).

CPN Tools provides a platform for editing, simulating, and analyzing CPN models in both academic and industrial fields. CPN Tools consists of two main components--a graphical editor and a backend simulator component. Using CPN simulator tools, the Petri nets engineer should be able to investigate property of Petri nets (i.e., firing of transitions) [20].

Therefore, it makes sense to extend AADLs with the analysis capability of CPNs to automatically simulate/analyze the dynamic behavior of AADLs.

This paper is structured as follows: the next section describes the approach to extend AADL's capability, section 3 describes the practicality of our method using an avionic system known as Ground Based Risk Mitigation System (GBRMS in sequel), section 4 discusses related work, and section 5 provides conclusions and future work.

2. METHOD: DYNAMIC BEHAVIORAL ANALYSIS OF AADL USING CPNS

Model-based engineering development utilizes formal modeling and state of the art toolsets to verify the behavior of a system so as to automatically generate code from specification and to generate test cases for system testing and validation [6][7]. Our framework supports the generation of formal models for specification and verification of both functional and nonfunctional requirements of a critical system by mapping AADL representation to Color Petri Nets (CPNs) [5]. More specifically, we have designed a simple set of mapping rules from AADL-to-CPN.

The justifications for using CPNs are very obvious. CPN is a wide-spread formal modeling language with very well established analysis results and supporting tools (editor, model checker, etc.). Color Petri Net models have been extensively used for specifying and analyzing complex, concurrent, distributed, asynchronous, parallel, deterministic, and non-deterministic systems. Using the analyzability capability of CPNs, such as reachability/coverability graph analysis, Petri net modelers are able to identify the presence of unsafe state (markings) that may jeopardize the safety of a system as a whole.

An important issue is the identification of a minimum subset (or core) of AADL components that can be faithfully mapped to the main elements of CPNs. Thus, proper decisions need to be made regarding what constitutes the subset of AADL and how this subset can be mapped to CPN for formal reasoning. For example, various types of interfaces and nesting rules used to create AADL

models may violate rules and principles governing systematic construction of CPN models.

Core AADL's elements include hardware and software components of AADLs such as processes, threads, bus, memory processors, etc. For the purpose of this work, AADL core elements consist of two sets of elements--active and passive components. Active elements are those elements having functional capabilities. Passive elements are those elements that are incapable of initiating or performing any task. Examples of active elements include threads and processes. Examples of passive elements include memory, buses, and devices.

Therefore, it makes sense to extend the capabilities of AADL-OSATE, a model-based engineering tool set, with CPNs to validate both the functional and nonfunctional features of the system. CPN models and their tool supports allow us to analyze behavioral aspects of a system. More specifically, by constructing a reachability graph obtained from AADL-CPNs models and using existing reduction techniques and tools, we should be able to analyze essential properties of a system design, such as verifying the absence of deadlocks and livelocks.

3. CASE STUDY: THE DEVELOPMENT OF GBRMS

A study has shown that ground and midair collision risk associated with the operations of Unmanned Aircraft System (UAS in sequel) may meet target levels of safety by integrating mitigation mechanisms into the system [16]. To this end, the goal of UAS-GBRMS is to provide a UAV operator with the level of awareness needed to prevent potential mid-air collisions among various aircraft (manned and unmanned) flying in the national airspace system (NAS).

In the simplest form, UAS-GBRMS is supposed to monitor the airspace in order to calculate the risk of mid-air collisions. The risk is computed using numerous parameters such as the number of aircraft, the capabilities of aircraft, and the characteristics of airspace together with a proper encounter model. The main components of the UAS-GBRMS system architecture (figure 1) include acquisition of real-time data provided by radars and ADS-B receivers in order to monitor the flight paths of UAV.

Several factors may jeopardize the safety of operations that are supported with the UAS-GBRMS. These factors include: a) possible overlap between flight paths of UAV and manned aircrafts (in sequel MAV) or other UAV; b) the UAV may simply run out of gas; c) mechanical limitation and weather conditions may result in inaccuracy about UAV flight path, unpredictable events, and traffic density. These factors may require a rapid course of action to be taken by UAVs in order to maintain the expected level of safety.

The UAS-GBRMS falls into the category of mission and safety critical systems because any deviation (minor or major) from functional and nonfunctional requirements can be easily translated into a fatal crash and loss of life. As such, the dependability, assurance methods and modeling, and standards (e.g., Do-178B) are mandated by various organizations (e.g., FAA) to guarantee a certain level of safety. The expected (or target level of) safety and reliability for avionic systems can be specified numerically using a possible range of values or max/min allowable values (e.g., 5 failures for every 7522 hours of flying or 664.7×10^{-6}). An example of other nonfunctional requirements (NFR) includes performance failures (end-to-end flow or communication failures).

Figure 1 shows the informal system architecture of the UAS-GBRMS that provides the required functionality. In this figure, radars constitute the primary sensors for collecting data about the UAVs and MAVs. The software architecture that provides full processing is composed of two blocks (dashed rectangular): 1) primarily radar capability and 2) command and control system which consists of a) the Data Fusion subsystem, b) the Risk mitigation subsystem, and c) the Display subsystem. In figure 1, a data link provides the required communication between radar system and command and control subsystems.

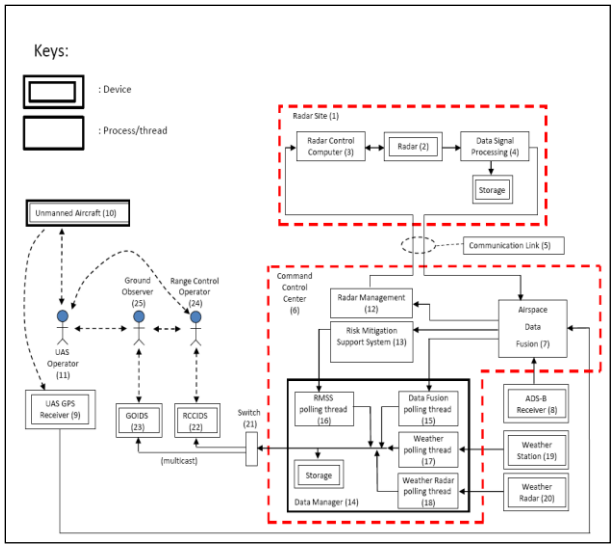


Figure 1. The Generic Architecture of UAS-RMS

The risk mitigation subsystem is responsible for computing the risk associated with operating in the NAS. The data fusion subsystem is responsible for combining and maintaining all data sent by various devices (e.g., Radar site). The display or visualization subsystem displays information to range and other UAV operators (and stores data). As can be seen in the figure, there are five input devices connected to the GBRMS: UAV-GPS (receiver device to collect UAV location from GPS), GPARS-Radars, Weather Radar, Weather Station and ADS-B Receiver (device to collect ADS-B data from participating aircraft). After processing in the GBRMS, all of the final data are sent to three output devices: the GOIDS (Ground observer interactive display system), RCCIDS Range Control Center Interactive Display System), and WXIDS (Weather Interactive Display System). The GOIDS is an interactive display of the GPAR-RMS for the UAV operator; the RCCIDS is interactive display of the GPAR-RMS for rang operator; the WXIDS is an interactive display for range control operator that displays weather radar data. Inside the GBRMS, the three subsystems talk to each other. Inside each subsystem, there are some processes and threads to process data and to transmit data or event information from/to each other.

Using the process discussed in the previous section, we first create an AADL specification of UAS-GBRMS from the informal one shown in figure 1. Once the AADL specification is constructed, we map the AADL specification to CPN according to the mapping rules (see table 1). Mapping is accomplished by replacing threads, processes, devices, etc., with the appropriate CPN notations according to table 1.

AADL Components	**Mapping**	**CPN Elements**
Ports (in/out data/event port, port group, event data port, data access)	⟺	place
System	⟺	Hierarchy transition (which includes the substitution transitions) or transition
Process	⟺	Hierarchy transition or transition
Thread	⟺	Transition
Device	⟺	Transition
Memory	⟺	Place
Bus (physical connection)	⟺	Flows
Connections	⟺	Flows
Processor	⟺	Transition

Table 1: Rules for mapping between AADL and CPN

The following example (figures 2 and 3) shows how to map AADL to CPN according to the mapping rules. The CPN places model the features part of threads, so the in data port *weather_radar_input* mapped to the place with the corresponding name. We do the same to the out data port *weather_radar_output*. CPN transition models the thread itself. The thread *weather_radar_polling* is mapped to transitions having the same name. Finally, arcs are used to model the connection.

Using operational scenarios documented during requirement analysis, we identify various states (i.e., initial marking) of the GBRMS by adding the required tokens and inscriptions to the CPN model in order to validate the dynamic behavior of our system.

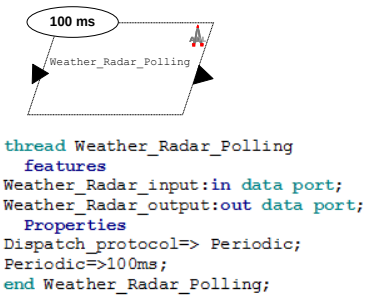


Figure 2: Example of AADL thread component

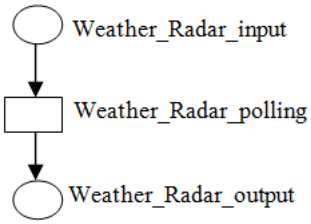


Figure 3: CPN graph transformed from AADL thread component

The following narrative informally describes an operational scenario where within the detecting range of the radar there is one MAV and one UAV. They both are flying at the same altitude and at the same speed but in opposite directions. The system receives the position of the UAV and MAV, namely (x_i, y_i, z_i) coordinates from the external devices (e.g., radar site). The coordinate values fused with other information (e.g., weather information) are used as the initial markings of the CPNs model. Using the values of x, y, and z, the system calculates the minim allowable separation distance between these planes needed for a possible mid-air collision to occur. In this case, the separation distances associated with a possible mid-air collision are <500 ft vertically and <1000 ft horizontally. With this, the system provides feedback to militate against mid-air collisions.

Figures 4-11 depict high- (abstract) and low- (concrete) level architectural and design specifications in AADLs and CPNs respectively. Figure 4 shows the high level architectural representation of the GBRMS system. In this figure the devices such as UAV, GPAR, and ADS-B Receiver provide data and events via output ports to the Airspace Data Fusion process. The weather radar sends data and events via an output port to the data manager process. This process, in turn, will send the result of analysis to the GOIDS The weather station sends its data via an output port to the weather radar polling thread, which then sends its analysis result to the WXIDS. The data polling thread sends the data gathered from other processes to the RCCIDS.

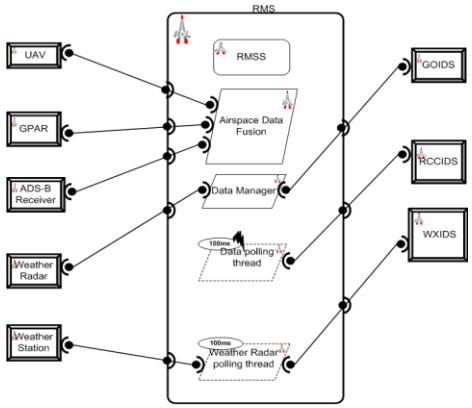


Figure 4: First Level Software View

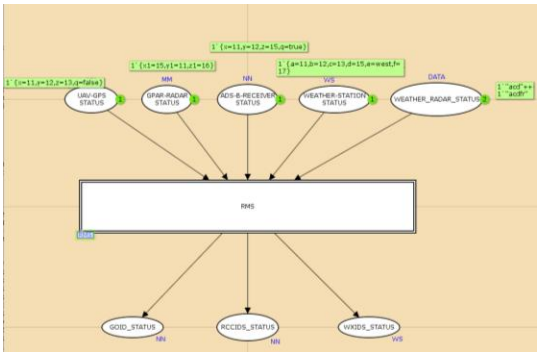


Figure 5: CPN Corresponding to Figure 4

Figure 5 shows the mapping from AADL to CPN. The transition RMS is a super transition corresponding to the system elements of AADL shown in figure 5 and hence can be hierarchically refined into another CPN or page (see figure 7). UAV-GPS STATUS, GPAR-RADAR STATUS, ADS-B-RECEIVER STATUS, WEATHER-STATION STATUS, and WEATHER-RADAR-STATUS are input places (or input data interfaces of the GBRMS system). Colorset NN is a record type that describes the positions or the coordination of the UAV; this information can be used to identify whether the UAV is flying within the detectable range of radar. Colorset MM is a

record type that holds the positions of an MAV within the range of radar. Colorset WS is a record type that includes weather information such as wind direction, temperature, etc. Color-set DATA is a string type that provides some information.

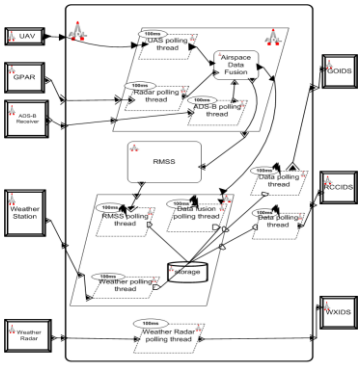


Figure 6: Second Level Software View

Figure 6 represents the concrete view of the GBRMS shown in figure 4. There are three main subsystems: RMSS, Data manager subsystem, and Airspace data fusion subsystem. The Airspace Data fusion subsystem performs the following tasks: fusing aircraft position data using optimal fusion algorithms; collects and maintains all of the data coming from the UAS polling thread, Radar polling thread, and ADS-B polling threads; sends the final data to the other two subsystems--namely the RMSS (Risk Mitigation Support System) and the display subsystem.

In figure 6 the RMSS computes risk computations. To this end, it receives data from the data fusion system it then sends the final result to the data manager process. The Data Manager Process is in charge of gathering all information and displaying this information to a range control or ground observer operator and stores data. There are three threads in the Data Manager Process: the data fusion polling thread, the RMSS polling thread, and the weather polling thread. These three threads send data to storage. The two different kinds of Data polling threads read the data from storage and send them to the RCCIDS and GOIDS respectively. The Weather Radar polling thread gathers external weather radar data from Weather Radar and sends them to WXIDS, which is an interactive display of external weather radar data for the range control operator.

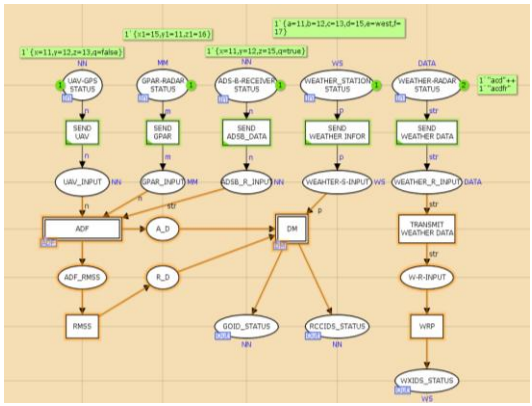


Figure 7: CPN Corresponding to Figure 6

The detailed design of the transition RRMS in figure 6, are illustrated in figure 7. In this figure the arc expression is added, where n is a variable of type NN, m is a variable of type MM, p is a variable of type WS, and str is a variable of type DATA. DM and ADF are another two hierarchical transitions that are replaced by the subpages illustrated in figures 9 and 11.

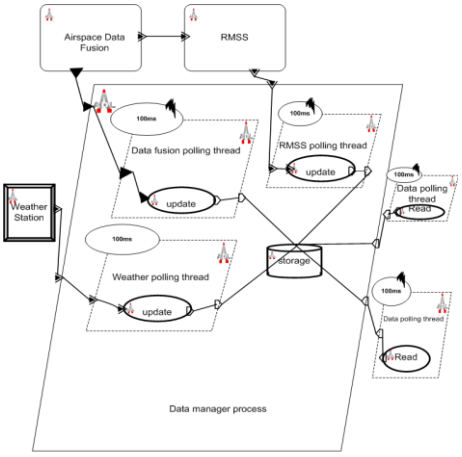


Figure 8: Third Level Software View-Data Manager Process

Figure 8 shows the detailed description of the Data Manager Process. The Data Manager Process receives comprehensive data regarding aircraft positions with a sporadic thread (the Data fusion polling thread), updates these data, and stores them in storage. The sporadic thread RMSS polling thread receives data from RMSS, updates data, and retains them in storage. The weather polling thread is a periodic thread. It receives data

from the Weather Station device, updates the data every 100 ms, and saves them in the storage. Two kinds of Data polling threads read the data from storage.

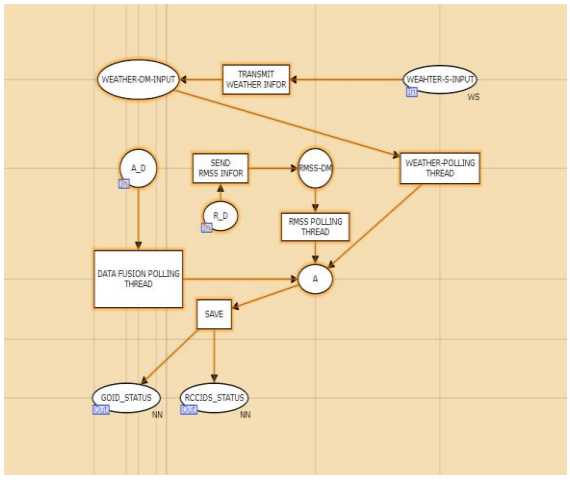


Figure 9: CPN Corresponding to Figure 8

Figure 9 shows the detailed design of the transition DM (i.e., Data Manger) where Place A represents the input data interface of storage. It connects to the transition SAVE, which performs the saving task. GOID-STATUS and RCCIDS-STATUS are output interfaces of data manager process. They access storage to get the data and then send them to different data polling threads.

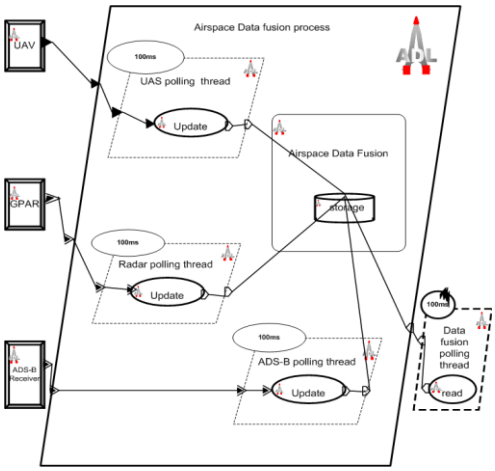


Figure 10: Third Level Software View-Airspace Data Fusion Process

Figure 10 shows the detailed description of Airspace Data fusion process. The periodic thread UAS polling thread receives data from UAV every 100 ms, updates the old information, and saves the new data in storage. The Radar polling thread and ADS-B polling thread perform the same set of tasks as Airspace Data.

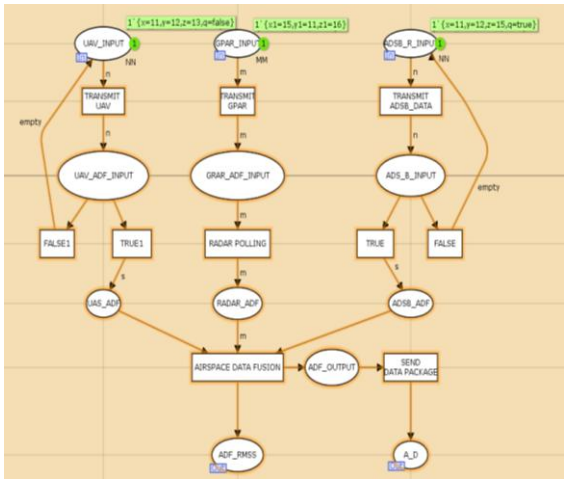


Figure 11: CPN Corresponding to Figure 10

Figure 11 shows the detailed design of the transition ADF. When a record is transmitted to UAV_ADF_INPUT, the system will make a decision according to the value of q. where q is Boolean where q=false means the absence of UAV; q=true means the presence of UAV. If a UAV is not within range of radar, transition FALSE1 will return empty to UAV_INPUT. Otherwise, the rest of the record is transmitted to the next place UAS_ADF. The same thing happens to the place of ADS_B_INPUT.

The CPN figures used in this paper were edited, simulated, and analyzed by using CPN Tools [15].

4. RELATED WORKS

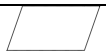
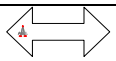
There have been some work to enhance the capability of an AADL-based notation and tool set; the tool set is named Ocarina/Cheddar [10] has been used to prototype and implement a distributed real-time system. The authors informally discussed the mapping between AADLs and CPNs.

In [18] the author proposes a general methodology for translating AADL to BIP, which focuses on threads, processes, and processors as well as the behavioral annex.
The work reported in [19] is yet another attempt for translating AADL into Petri nets.

5. Conclusion and Future Work

In this paper we have shown how AADL specifications can be extended by using CPN modeling notations to simulate the dynamic behavior of a system called the GBRMS. In order to extend AADL with CPNs behavioral analysis and specifications, we devised a simple set of mapping rules to create the structure of CPN. We used operational scenarios documented during requirements engraining to extract initial marking and CPN inscriptions. These are used to describe dynamic aspects of CPN--namely token games. We applied our method in the development of the GBRMS. The mapping of AADL-to-CPN is performed manually. Our ultimate goal is to automate the behavioral verification of an avionic software system known as GBRMS. Currently we are developing an Eclipse [11] plug-in to fully automate the mapping from AADL to CPN. To this end, we are investigating the feasibility of model driven and model transformation techniques to map AADL-XML representation of a system into Petri Nets Mark-up Language (PNML), which is a standard interchange format for Petri net tools. The main goal of the ISO/IEC 15909 [17] standard is to promote the extensive acceptance of different classes of Petri nets..

Appendix A: AADL

AADL Concepts	AADL DIAGRAM	AADL DESCRIPTION
SYSTEM		Represent composite or hierarchical structure of a components
PROCESS		a protected virtual address
THREAD		Represents schedulable unit of concurrent execution of code
THREAD GROUP		Represents Collection of threads
DATA		Represents sharable data
SUBPROGRAM		Represents Callable unit of sequential code
MEMORY		Represents data storage retaining data values
BUS		represents communication channels among hardware elements

Appendix A: AADL (continued)

DEVICES			Represent external components such sensors and radars
CONNECTIONS			Represent Connects ports in the direction of their flows
PORTS	DATA PORT		Represent input and output interface
	EVENT DATA PORT		
	DATA ACCESS		
PORTS GROUPS			Represents a collections of input/out places

Appendix B: CPN

Formal Definition: A Colored Petri Net is a nine-tuple CPN = (Σ, P, T, A, N, C, G, E, I) *where*

- Σ= a finite set of non-empty types,
- P = a finite set of places,
- T = a finite set of transitions,
- A = a finite set of arcs (or flows) connecting places to transitions or transitions to places,
- N = a node function and is defined from A into $P \times T \cup P \times T$.

- C = a color function. It is defined from P into Σ,
- G = a guard function,
- E = an arc expression function,
- I = an initialization function.

ACKNOWLEDGMENT

This work was partially funded by the United States Air Force under Contract Number FA4861-07-R-C003.

REFERENCES

[1] P. FEILER, D. CLUCH, J. HUDAK. THEARCHITECTURE ANALYSIS & DESIGN LANGUAGE (AADL): AN INTRODUCTION. CMU/SEI-2006-TN-011.

[2] M. SHAW, D. GARLAN. SOFTWARE ARCHITECTURE: PERSPECTIVE ON AN EMERGING DISCIPLINE, PRENTICE HALL 1996.

[3] P. FEILER, B. LEWIS, AND S. VESTAL, "IMPROVING PREDICTABILITY IN EMBEDDED REAL-TIME SYSTEMS," CARNEGIE MELLON SOFTWARE ENGINEERING INSTITUTE, CMU/SEI-2000-SR-011, OCTOBER 2000.

[4] G. BOOCH, J. RUMBAUGH, I. JACOBSON. THE UNIFIED MODELING LANGUAGE USER GUIDE. ADDISON WESLEY, 1999.

[5] K. JENSEN. COLORED PETRI NETS: BASIC CONCEPTS, ANALYSIS METHODS AND PRACTICAL USE VOL.1, 2ND EDITION, SPRINGER, 1997.

[6] S. VESTAL. METAH PROGRAMMER'S MANUAL, VERSION 1.22. HONEYWELL TECHNOLOGY CENTER, METAH MATERIAL IS AVAILABLE UPON REQUEST.

[7] M. UTTING, AND B. LEGEARD. PRACTICAL MODEL-BASED TESTING: A TOOLS APPROACH. MORGAN KAUFMAN. 2007.

[8] M. BARBACCI, AND C. WEINSTOCK. MAPPING METAH INTO ACME. SPECIAL REPORT CMU/SEI-98-SR-006.

[9] N. MEDVIDOVIC, AND R. TAYLOR. A CLASSIFICATION AND COMPARISON FRAMEWORK FOR SOFTWARE ARCHITECTURE DESCRIPTION LANGUAGES. IEEE TRANSACTION ON SOFTWARE ENGINEERING, 2000.

[10] J. HUGUES, B. ZALILA, AND L. PAUTET. FROM THE PROTOTYPE TO THE FINAL EMBEDDED SYSTEM USING THE OCARINA AADL TOOL SITE. IEEE TRANSACTIONS ON EMBEDDED SYSTEM. VOL.7, NO.4. JULY 2008.

[11] D. CARLSON. ECLIPSE DISTILLED. ADDISON WESLEY, 2005.

[12] OSATE UPDATE SITE AT SEI: HTTP://LA.SEI.CMU.EDU/AADL/CURRENTSITE/RELEASENOTES-PREVIOUS.HTML.

[13] OBJECT MANAGEMENT GROUP (OMG). HTTP://WWW.OMG.ORG/.

[14] KURT JENSEN. A BRIEF INTRODUCTION TO COLOURED PETRI NETS. COMPUTER SCIENCE DEPARTMENT, UNIVERSITY OF AARHUS. NYMUNKEGADE, BLDG. 540, DK-8000 AARHUSC, DENMARK

[15] CPN-TOOL: WESTERGAARD.EU/WP CONTENT /UPLOADS/2010/09/CPN-TOOLS.PDF.

[16] R. WEIBLE, R. HANSMAN. AN INTEGRATED APPROACH TO EVALUATING RISK MITIGATION MEASURES FOR UAV OPERATIONAL CONCEPTS IN THE NAS. AIAA'S 4TH INFOTECH@AEROSPACE CONFERENCE 26-29, 2005.

[17] A PRIMER ON THE PETRI NET MARKUP LANGUAGE AND ISO/IEC 15909-2
L. HILLAH, E. KINDLER, F. KORDON, L. PETRUCCI AND N. TRÈVES. PETRI NET NEWSLETTER 76:9--28, OCTOBER 2009.

[18] M. CHKOURI, A. ROBERT, M. BOZGA, J. SIFAKIS. TRANSLATING AADL INTO BIP- APPLICATION TO THE VERIFICATION OF REAL-TIME SYSTEMS. IN PROC. OF MODELS ACES-MB-MODEL BASED ARCHITECTING AND CONSTRUCTION OF EMBEDDED SYSTEMS, 2008.

[19] BERTHOMIEU, B., BODEVEIX, J.P.,CHAUDET, C., DAL-ZILIO, S., FILALI, M., VERNADAT, F.: FORMAL VERIFICATION OF AADL SPECIFICATIONS IN THE TOPCASED ENVIRONMENT. IN: ADA-EUROPE'09. LNCS, VOL. 5570. SPRINGER (2009).

[20] K. H. MORTENSEN. EFFICIENT DATA-STRUCTURE AND ALGORITHMS FOR A CPNS SIMULATOR. 3RD WORKSHOP AND TUTORIAL ON PRACTICAL USE OF COLORED PETRI NETS AND CPN TOOLS, 2001.

Software Maintenance Supported by Refactoring

Gustavo Villavicencio[1]
[1]Facultad de Matemática Aplicada
Universidad Católica de Santiago del Estero
Campus de la UCSE, 4200 Santiago del Estero, Argentina
gustavov@ucse.edu.ar

Abstract— *In this paper a new maintenance scenario is outlined based on refactoring techniques. Specifically, refactoring techniques are classified according to two opposed program properties: understanding and efficiency. Understanding oriented refactoring disassembles the program preparing it for maintenance, whereas those oriented to efficiency rearrange it for running. Also, we show the challenges raised from this new perspective on maintenance. At present, this ongoing research is being carried out in the functional setting.*

Keywords: Program comprehension, forward refactoring, reverse refactoring, functional programming, maintenance

1. Introduction

Program comprehension is the process by which programmers build *mental models* of the program using their previous knowledge on programming and information captured from the source code. Such models reflect the understanding obtained by the programmer from the program, usually to carry out maintenance activities. The construction of mental models is supported by *cognitive models* that describe the information structures in the programmer's head and the way they are processed [1].

Many methods and tools based upon cognitive theory have been developed to assist programmers during the comprehension process. A common factor of these methods and tools we want to emphasize is that they are *not invasive*: the source code remains unchanged throughout the process. In this way, once the comprehension process has been completed, the maintenance activities are performed on the unchanged source code, which might be one cause for which software complexity increases over time: the software component is not effectively disassembled for maintenance. Thus it is performed on the current running system. Such system, even though efficient, is complex and monolithic due to the fact that it has been previously submitted to successive maintenance.

If we observe how other engineering disciplines perform maintenance on their artefacts, we will see that software engineering lacks an effective *disassembly* of the software artefact to perform maintenance properly. Disassembly is the way these engineerings, like aerospace, cope with the monolithical aspect and complexity of their artefacts. So, what is the "screwdriver" in software engineering? In this paper we argue that this role can be played by *refactoring*, which is defined as a technique to transform the internal structure of a software system preserving its behaviour [2].

In this short paper, we outline a new maintenance scenario based on refactoring. Specifically, refactoring techniques are classified based upon two usually opposed program properties: *efficiency* and *understanding*, being the former useful for running and the latter for maintenance.

The rest of the paper is organized as follows: in section 2 we review the current role of refactoring as a mechanism to make subsequent maintenance easier. In section 3 we account for the classification proposed for refactorings. Section 4 describes the new maintenance scenario outlined by such classification and the challenges that it raises. Finally, section 5 shows the conclusions.

2. Refactoring in the Comprehension Process

In the imperative setting, the only reference we have found to refactoring techniques as a mechanism to carry out program comprehension was in [3]. Additionally, in some literature [4], [5] the use of refactoring as a way to improve the internal structure of a program and thus make future maintenance easier has been reported. However, we have some objections to this *preventive understanding* based on refactoring:

- Opposed to [4], [5], other literature has reported results that might question the efficiency of such preventive understanding. Among other works, [6], [7] have reported that efficiency and understanding are two opposite program properties. Therefore, trying to improve comprehension of a running program for making later maintenance easier can damages efficiency.
- On the other hand, it is not clearly documented what kind of refactorings are carried out, but certainly it is not a refactoring for disassembling a program.
- Moreover, details on the quantification of the improvement gained by this preventive comprehension based on refactoring have not been found. Furthermore, lack of

quantification on the benefits of refactoring is beyond this particular application [8].

Therefore, we believe that preventive understanding based on refactoring is a "shallow understanding" and it is not oriented to disassemble the source code.

In [9] we have begun to draw an alternative way to perform program understanding on functional programs based on refactorings. Further, in [10] we propose a categorization of refactoring techniques based on two opposed program attributes: efficiency and understanding. The classification is proposed in such a way that a refactoring placed into one category has an inverse in the other one. It allows to go in both directions, exactly like a screwdriver.

3. Forward and Reverse Refactorings

Underlying each refactoring is the purpose of improving some aspect of the program. We argue that the two main aspects for improving a software system are efficiency and understanding. On the basis of such properties, we propose a classification of refactorings. The set of refactorings oriented to improve efficiency are named *forward refactorings*, and those oriented to improve comprehension are named *reverse refactorings* [1]. The first ones are denoted by $\overleftarrow{r_1}, \ldots, \overleftarrow{r_n}$ and the latter by $\overrightarrow{r_1}, \ldots, \overrightarrow{r_n}$. In the sequel we describe some of these refactorings:

Accumulation parameter introduction / Accumulation parameter removal:
With the ACCUMULATION PARAMETER INTRODUCTION technique, the original and inefficient definition of a functional program is generalized by introducing an extra parameter which is used to accumulate additional information during a computation [12]. It is one of the most typical techniques used to improve functional program efficiency. On the contrary, ACCUMULATION PARAMETER REMOVAL makes a function definition clearer by eliminating the non-inductive parameter. Basically, this refactoring entails the replacement of the accumulation parameter by a function performing the calculation of such value. For this purpose, the function receives as parameters an element from the inductive data structure and the output of the recursive function. To meet the last requirement, the recursive function calls the new function, placing as one of their arguments a call to itself. Note that the separation of the function replacing the accumulation parameter from the recursive function is a kind of chunking since the new function involves sentences performing a specific task. It is not a trivial refactoring and thereby it is very unlikely that it will be fully automatized.

[1]These terms have been renamed with respect to [10] in order to "synchronize" with the terminology in [11], where *forward engineering* and *reverse engineering* have been defined.

Inlining (unfolding) / Folding: INLINING is a widely known technique for compilers to gain efficiency. Basically, by this technique a function call is replaced by the body of the function, and thereby saving execution time and memory resources. In the source code level, a modification on this kind alters the organization of the code, going toward a monolithic style. Going into the opposite direction, i.e. applying folding to gain understanding, involves the detection and replacement of the right-hand side of an identified function with a call to that function. Viewed from the perspective of styles of programming, this technique entails decomposing the code in smaller fragments (objects, procedures, functions, etc) which resembles the divide-and-conquer strategy to cope with problems. Hence, the code fragmentation resulting from folding refactoring is similar to chunking the code in a bottom-up program comprehension process.

Function specification / General function abstraction:
Although there is no consensus on source code clone definition, it might be considered as copies or near-copies of other portions of code, often created by copying and pasting portions of source code [13]. Specifically, GENERAL FUNCTION ABSTRACTION is a refactoring that abstracts expressions within a *clone class* into a call to a new function [14]. Inversely, the forward refactoring instances the clone class generating a specific clone. We must observe that, like in FOLDING refactoring, the purpose here is to rise up an abstraction from similarities or relations detected among sentences in the source code. The difference is the nature of the similarities or relations. In FOLDING refactoring, grouping sentences is based on the need to dismiss complexity, so that each group performs specific tasks. That is, there are data and control dependencies among sentences. On the contrary, the class clone obtained by GENERAL FUNCTION ABSTRACTION refactoring, groups sentences which do not necessarily have data and control dependencies among them, and perhaps, similarities are only syntactical.

Tupling (merging) / Splitting: TUPLING is a well-known transformation tactic to improve program efficiency by grouping recursive functions into a tuple, avoiding multiple traversal over the same data structure. Conversely, SPLITTING refactoring works by selecting an element of the tuple and working out everything needed to calculate that element. The calculated dependencies are then simply extracted and isolated from the rest of the function [15]. Specifically, in the HaRe refactorer this refactoring is based on *slicing techniques* [16]. Again here, the interleaved code is untangled to decrease complexity. But differently, the mechanism to carry

out this process is automatic, based on data analysis and control dependencies, i.e. slicing. As in the other refactorings, here we have other mechanism to group sentences under an abstraction, i.e. we are chunking the code.

Fusion / Reforestation (fission): Given a composition $f(g\ x)$, $g\ x$ yields an intermediate data structure which is consumed by f. The intermediate structure allows modularity but affects efficiency. Fusion (deforestation) technique [17] attempts to avoid intermediate data structure generation composing both function into one $h(x)$ that has the same semantics as $f(g(x))$ but does not require the intermediate structure. Applying FISSION on a complex monolithic optimized program, one constructs a simpler, more modular specification or prototype, identifying the components from which the complex program might have been assembled [18]. As we can see, this refactoring also leads to chunk the code.

This unfinished catalog contains reverse refactorings identified from their forward refactoring counterparts, i.e. $\overrightarrow{r_i} = \overleftarrow{r_i}^{-1}$. Obviously, the application of forward refactorings improves efficiency but damages understanding, and conversely the application of reverse refactorings improves understanding but damages efficiency.

4. A New Maintenance Scenario

The refactoring classification based upon efficiency and understanding spawn a new scenario to carry out maintenance activities (figure 1). In some point during the running of the system a maintenance requirement that triggers a comprehension process may arise. After obtaining a precise understanding of the source code functionality by reverse refactoring (plus calculation in case of [10]) the maintenance activities are carried out. Then, the system should be run again by performing the opposed refactorings to those performed at the beginning, i.e. forward refactorings. In this context a software component can be viewed from a new perspective

> *A software component is a device with a switch: In on, it is running and in off it goes to maintenance by* ***automatic reverse refactorings*** *(comprehension direction). After maintenance, the component is switched on again and restored to running by* ***automatic forward refactorings*** *(efficiency direction). That is, reverse refactorings "decompose" the system preparing it for maintenance as forward refactorings "rearrange" the system preparing it for running.*

The previous description represents an ideal scenario. However, there are some issues arising from this context:

1) By applying a sequence $\overrightarrow{r_1}, \ldots, \overrightarrow{r_k}$ (with $k \leq n$, i.e. not all reverse refactorings can be applied), successive and semantically equivalent versions of the source code are obtained: $\overrightarrow{v_1}, \cdots, \overrightarrow{v_k}$. Ideally, $\overrightarrow{v_k}$ should be the one that best satisfies the comprehension requirements, but not necessarily the one that satisfies the maintenance requirements. So, given a maintenance specification, which is the best $\overrightarrow{v_i}$ with $1 \leq i \leq k$ on which to apply such specifications? This question opens the road to other ones:
   a) which is the best way to match system functionality against maintenance specification in order to detect the best $\overrightarrow{v_i}$?,
   b) which is the most appropriate source code representation to match source code with maintenance specification?,
   c) which is the most appropriate maintenance specifications representation to perform this process?,
   d) how can we measure the optimal result?.
2) After applying a sequence $\overrightarrow{r_1}, \ldots, \overrightarrow{r_k}$ and carrying out maintenance, can we restore the system to its normal course by performing $\overleftarrow{r_k}, \ldots, \overleftarrow{r_1}$, or a subset of this sequence?. Alternately, is there a way to carry out maintenance on the source code that guarantees that by performing the sequence $\overleftarrow{r_k}, \ldots, \overleftarrow{r_1}$ we can restore the system to its normal course?,
3) Clearly, the refactoring machinery plays a critical role in this context and, therefore, must be submitted to constant improvement. So, for example, any new detected refactoring in any direction must be analyzed and its reverse developed.

The issue arised by item 2 may be the most challenging and can be investigated by means of *maintenance patterns* [19]. These consist of *roles* that can be bound to program entities and a set of *constraints* that these bindings must satisfy. In our context, the roles would be the parts of the artefact in which it is disassembled whereas the constraints would be the relations among them. So, during maintenance constraints should be observed to guarantee that the program can be restored to its normal course by forward refactorings. In the same way, item 3 can be investigated following the work developed in [20]. Regarding issues in item 1 further investigation must be undertaken to gather more insight.

5. Conclusions

As it has been previously argued, refactoring (more precisely reverse refactoring) is another way to perform program comprehension, different from the traditional ones (bottom-up, top-down and hybrid) which do not alter the structure of the source code. Refactoring is a more "invasive" technique that has the additional advantage of improving the conditions under which maintenance is carried out. The traditional program comprehension strategies only provide an explanation

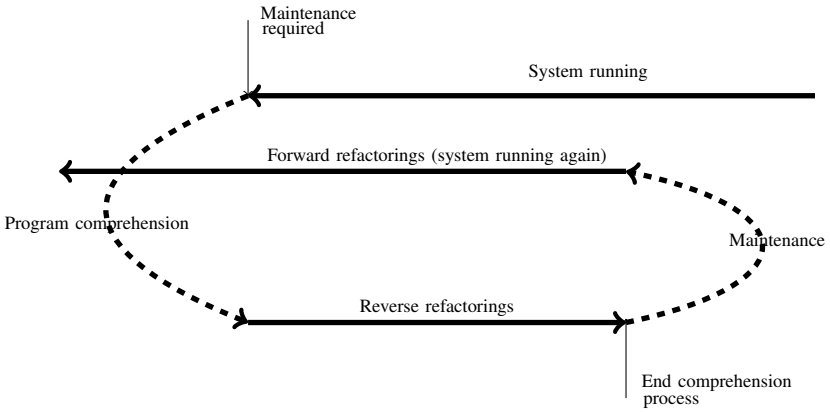


Fig. 1: Maintenance scenario based on reverse and forward refactorings

of what the program do, but maintenance is applied to the complex code making it even more incomprehensible. Reverse refactoring provides the opportunity to perform maintenance on a well-structured code.

An important observation we should emphasize is that the proposed refactoring classification gives rise to a new view of what a software component is. That is, a software artefact can now be effectively disassembled by reverse refactorings and the maintenance activities performed on this scenario. After that, the artefact should be rearranged by forward refactorings to take it back again to its natural environment. That is exactly the way other engineering disciplines work. This view entails the development of a new kind of methods, techniques and tools devoted to software maintenance.

Although our exploration of refactoring techniques as a new way of carrying out program comprehension has been conducted on the functional setting, we believe that the same results (specially the classification of refactoring techniques) can also be obtained on the imperative setting.

The main contribution of this paper is not to provide solution to any particular problem. Instead, we are trying to call attention of the scientific community to the fact that we have solid and well-founded evidence that we are standing in the doorway of a totally new way of carrying out maintenance on software systems on the basis of forward and reverse refactorings.

References

[1] M.-A. Storey, "Theories, tools and research methods in program comprehension: past, present and future," *Software Quality Control*, vol. 14, no. 3, pp. 187–208, 2006.

[2] M. Fowler, "Refactoring home page," `http://www.refactoring.com`, 2010.

[3] B. D. Bois, S. Demeyer, and J. Verelst, "Does the "refactor to understand" reverse engineering pattern improve program comprehension," in *Ninth European Conference on Software Maintenance and Reengineering (CSMR'05)*. Manchester,UK: IEEE, March 2005.

[4] M. Arora, S. S. Sarangdevot, V. S. Rathore, J. Deegwal, and S. Arora, "Refactoring, way for software maintenance," *International Journal of Computer Science Issues*, vol. 8, no. 2, March 2011.

[5] T. Mens and T. Tourwé, "A survey on software maintenance," *IEEE Transactions on Software Engineering*, vol. 30, no. 2, pp. 126–139, February 2004.

[6] Z. Hu, T. Yokoyama, and M. Takeichi, "Optimizations and transformations in calculation form," in *Summer School on Generative and Transformational Techniques in Software Engineering (GTTSE'05)*, ser. LNCS, R. Lämmel, J. Saraiva, and J. Visser, Eds., vol. 4143. Braga, Portugal: Springer-Verlag, July 2006.

[7] M. A. Tullsen, "Path, a program transformation system for haskell," Ph.D. dissertation, Yale University, May 2002.

[8] S. Bryton and F. B. e Abreu, "Modularity-oriented refactoring," in *12th European Conference on Software Maintenance and Reengineering*. Athens, Greece: IEEE, April 2008.

[9] G. Villavicencio, "Refactoring for comprehension," in *Draft Proceeding of the 8th. Trends in Functional Programming*. New York, USA: Seaton Hall University, April 2007.

[10] ——, "A bottom-up approach to understand functional programs," in *Fourth International C* Conference on Computer Science & Software Engineering (C3S2E 2011)*. Montreal, Quebec, Canada: ACM, May 2011.

[11] E. Chikofsky and J. H. C. II, "Reverse engineering and design recovery: a taxonomy," *IEEE Software*, vol. 7, no. 1, pp. 13–17, 1990.

[12] O. D. Moor and G. Sittampalam, "Generic program transformation," in *Proc. 3rd International Summer School on Advanced Functional Programming, LNCS 1608*. Braga, Portugal: Springer-Verlag, 1998, pp. 116–149.

[13] A. Lakhotia, J. Li, A. Walenstein, and Y. Yang, "Towards a clone detection benchmark suite and results archive," in *11th IEEE International Workshop on Program Comprehension, IWPC'03*. Portland, Oregon, USA: IEEE, 2003.

[14] C. Brown and S. Thompson, "Clone detection and elimination for haskell," in *PEPM'10*. Madrid, Spain: ACM, January 2010.

[15] C. Brown, "Tool support for refactoring haskell programs," Ph.D. dissertation, University of Kent, Canterbury, Kent, England, September 2008.

[16] M. Weiser, "Program slicing," in *Fifth International Conference on Software Engineering, San Diego*, California, March 1981.

[17] P. Wadler, "Deforestation: transforming programs to eliminate trees," *Theoretical Computer Science*, vol. 73, pp. 231–248, 1990.

[18] J. Gibbons, "Fission for program comprehesion," in *8th Int.Conference on Mathematics of Program Construction*, ser. LNCS, T. Uustalu, Ed., vol. 4014. Kuressaare, Estonia: Springer-Verlag, July 2006.

[19] I. Hammouda and M. Harsu, "Documenting maintenance tasks using maintenance patterns," in *Proceedings of the Eighth European Conference on Software Maintenance and Reengineering*. Tampere, Finland: IEEE, March 2004.

[20] K. Prete, N. Rachatasumrit, N. Sudan, and M. Kim, "Template-based reconstruction of complex refactorings," in *26th IEEE International Conference on Software Maintenance*. Timisoara, Romania: IEEE, September 2010.

Towards Intermediate-Agile Model Based On Agile Through Requirement Management And Development Enhancements

Amr Rekaby[1], and Maha Soliman[2]
[1] HP Egypt
[2] Helwan University, Cairo, Egypt
rekaby0@hotmail.com, maha.m.soliman@gmail.com

Abstract - *the requirements management and development are very important parts and activities in the software development life cycle, and the "Agile" is one of the most common and widely used software life cycle models. But applying agile methodology most probably shows that, it contains some problems related to requirements management and development. This paper suggests a new software development life cycle model called "Intermediate-Agile" based on "Agile" model but with some changes in the life cycle, best practices, documentation templates and enhancements in the activities of requirement gathering and management to cover these problems.*

After explaining "Intermediate-Agile" and "Agile" models, the paper evaluates the new model against normal agile model according to the CMMI-Dev 1.2v goals. CMMI is a quality assurance standard which contains many quality areas targeting the best quality of the software cycle deliverables. This paper focuses on "requirement management" and "requirement development" process areas in CMMI which are the most areas that "Intermediate-Agile" model represents the enhancements.

Keywords: Software Requirements, Requirements Management, Requirement Development, CMMI, Agile, intermediate-Agile.

1 Introduction

While software requirements gathering, maintaining, promoting to next phases and other requirements related issues are common reasons of project failure, delivery delay, conflicts and non-user satisfaction of the final product, So this paper focus on this phase of the software development life cycle to add more enhancements in this phase to avoid these problems as much as possible.

Due to practical situations, Agile is a very common model [4] but it initiates some problems related to requirement management [1], these problems are the motivation source for this paper to focus on "agile" model analyze it then present a new model called "Intermediate-Agile" (I-Agile for simplicity) model which is based on agile basically but with enhancements in the requirements phase and requirement development activities in the agile model. I-Agile model also present a different flow of the phases in the life cycle as shown in the next sections. After showing the agile model and the Intermediate-Agile model, we need criteria to can evaluate the enhancements which are added by the Intermediate-Agile model, so the CMMI common improvement process is chosen to be the evaluation criteria.

CMMI contains many process areas, but in this paper we focus on "Requirements Management" and "Requirements development" process areas and their goals and practices as shown in CMMI sections.

The flow of the paper is: showing Agile model, presenting new Intermediate-Agile model, present CMMI generally then discuss the CMMI process areas were chosen as evaluation criteria's, after that make the practical evaluation of two models versus CMMI goals to show which fit CMMI better, then the conclusion of this paper.

The fitting of the CMMI is not the target of software industry, but CMMI introduces the best practices and activates in the software development. So the best fitting into CMMI standards is the best in the development workflow success, in final product quality, in schedule fitting and so on.

2 Iterative and agile life cycle models

2.1. Software development life cycle models introduction

Software development life cycle contains many phases (steps) to can establish the final product to satisfy the user needs. Software development life cycle models are these methodologies that construct the life cycle, Activities, roles, duties, and phases… of the product construction.

2.2. Iterative Development Model (General view)

This model depends on making the product in incremental way, by dividing the product into smaller parts and building them as separate deliveries. Starting by doing the analyzing of current iteration, design, implement, deploy, test, evaluate it and then return again to analyze the next iteration. At the end of the iteration, the team should provide a deliverable package to the customer (as far as the project achieve) and a feedback from customer should be collected.

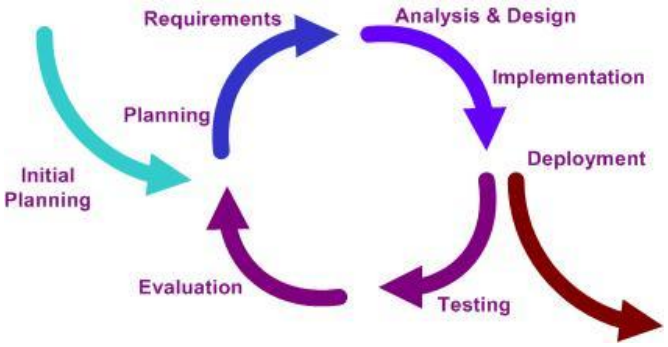


Fig.1.Incremental Development Model

2.3. Agile Model

Agile is a special case of Incremental Model. Agile ensure that the iteration period have to be from two weeks to four weeks. Agile also request that the deliverable component to the customer each phase should be executable components which putted under verification and evaluation from the customer side at the end of the iteration. The feedbacks from customer become very important input to the next phase. This Idea makes the product path always not far from the customer needs. The customer feedback always gives the schedule of the project the reliability because when the project management gets a satisfaction of part after part of the project during the time so the project is on the correct path to the success delivery. Agile is always decreasing the amount of documents and the details in it as much as possible to be able to have an output in the iteration period (two to four weeks). Agile model by definition requests that the team should seat in the same location and recommend being in customer side to be able to overcome the shortage in documentation by direct communication between team and the customer's interface. In agile model, before starting the first iteration there is a phase occurred once "initial planning" which have some activities such as project scope definition, abstract project architecture designing, abstract project planning and scheduling. In some projects there are some use cases details and dependencies don't appear in use-case document and depends on the team communication and the agile environment for knowledge sharing. The big issue that faces agile is that, the requirements are not in deeply discussed in the beginning of the project, which could raise a problem in later iterations with some requirements that will destroy the base design that happened in early iterations. These problems affect the final product quality, and for sure affect the delivery dates and budgets [5].

3 Proposed Intermediate Agile Model

This paper proposes a new software life cycle called Intermediate-Agile (I-Agile). This new model is based on normal common agile model but with changes in life cycle, best practices and documentation templates. This section presents the main points proposed in the new I-Agile model.

- The first change exists in the life cycle itself. I-Agile model introduces a phase called "Initial Requirement" to be before the first "iteration requirement" phase. This "Initial Requirement" phase should take 30% to 40% of the total requirements analysis effort in the project, to accurately verify the scope and cover each part even if with a high level touch. This will make each iteration requirement phase shorter than the equivalent in agile model.

- Afterwards the initial requirement phase should cover in details all requirements related to the first iteration (minimum the first iteration). This will make each normal iteration requirement phase focuses on the further iterations requirements. In first iteration, the iteration requirement phase should care about the second or even later phases in details. This step in advance gives the project the chance to find any conflict between requirements without reworking time and effort in something that will be revisited again. This means that there is no extra effort will be added in I-Agile model, just it re-organizes the effort. The requirements analysis effort will be the same like normal Agile model, but I–Agile model will consume 30-40% of these effort in this proposed phase, and make each iteration requirement phase is shorter than itself in agile model. Common agile model contains "Iteration-0 initial requirement" phase. This phase considers building the team, system initial architecture, setting up the environment and project estimation. These activities are exist also in I-Agile model in the same phase, but this common agile phase is not enough in many projects to build the base understanding which make the requirements easy to be handled in the future iterations. So I-Agile model suppose the "initial requirement" phase which should cover this point as described earlier. I-Agile model is shown in figure 2.

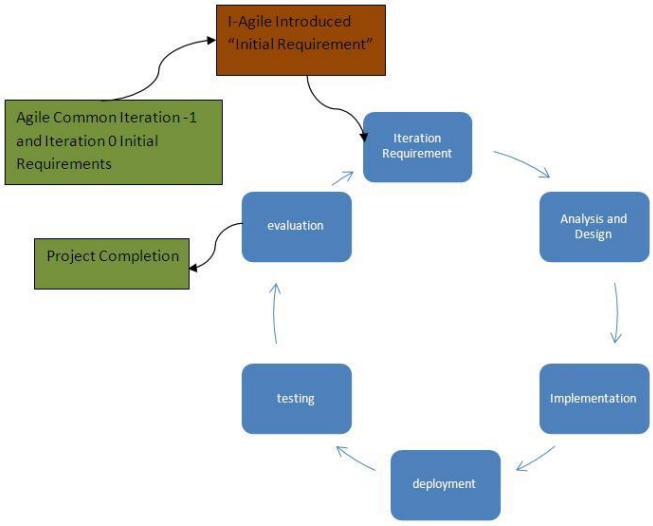


Figure 2: Intermediate-Agile Model

- Another best practice that is proposed by I-Agile, a template of documentation that should be filled during the initial requirement phase, this document template covers all the use cases in the system but with high level perspective. This template contains the following information:

  - Use case name (Identifier): this is the name of the use case and also handled as unique identifier for it.

  - Short description: this is a description abstractly for the use case and its target from it, without steps, just description paragraph.

  - Owner component: map this use case to which component responsible for it.

  - Related components: list the related components will be used during this use case, such as authorization component.

  - Related use cases: list the use cases which will be used from this use case.

This template shows the dependency between businesses which make the future tracking of any change easier.

Another recommendation from I-Agile is that, the UML architecture and components model which presents the dependency between components is mandatory to close the initial requirements phase. The architecture model always contains the structure of the project from layers and technologies view. In some cases in agile projects the component model doesn't take the enough interest. Intermediate-Model recommends the component model as a mandatory document because this model plus the simple use case form shown above will help the team a lot to don't go into conflict or confusing.

4 Experiments and Results

CMMI is one of the most common and reliable process improvement approach and standards of product quality evaluation [2]. The paper provides a measurements which compare I-Agile versus normal Agile model against CMMI standards and goals.

The paper choices CMMI process areas to be the referee in the evaluation between agile and Intermediate-Agile models. Each CMMI process area monitors the organization from different views such as "Project planning" which is interested in project management activities, specifying start and end dates for different phases, different phases dependencies on each other and so on.

Each process area is composed of specific goals and generic goals. Specific goal is a goal that is specifically related to this process area. For example in Process area "Requirement management" there is a specific goal called "Manage requirements", this specific goal is related to this process area, and doesn't related to other process areas and it focuses on how to manage project's requirements, update requirements with changes, specify inconsistencies between project plan and products and so on. While generic goal is a goal which is related to all process areas and that's why they are called generic (generic goals are out of this paper scope).

As shown in figure 3 each specific goal contains some specific practices that should be accomplished in order to achieve this specific goal and the same for generic goals contains generic practices.

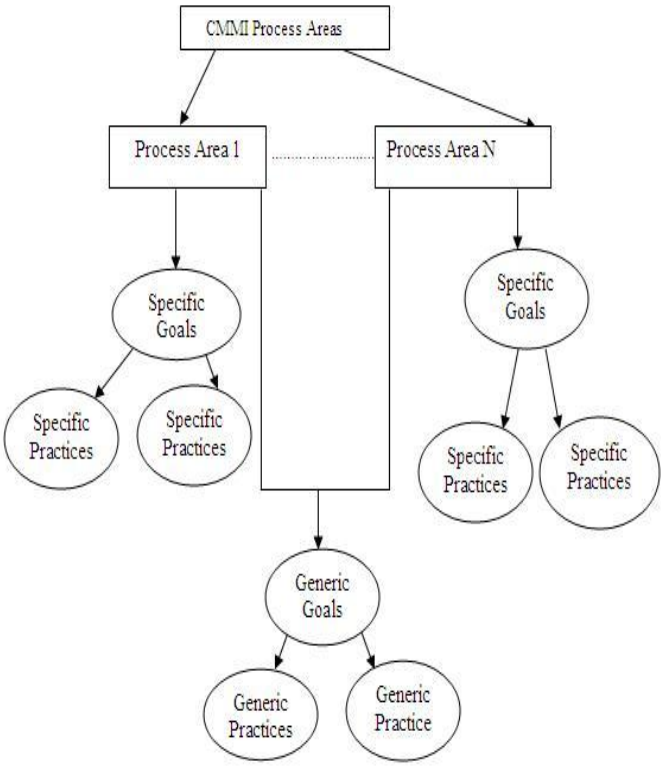


Figure 3: CMMI process areas structure

While the enhancements that are presented in this paper are regarding requirements, so the only CMMI process areas that are involved are "Requirement Management" and "Requirement Development" areas. The results come from evaluating real projects that use Agile and others use I-Agile model according to each CMMI goal in the involved process areas.

In practical experiments, real projects applied part of these papers' recommendation recognize the effect of these changes on the product quality. The actual projects that do that are around 10 projects between medium size projects (from 3000 to 5000 working hours) and small projects (from 1000 to 2000 working hours).

After applying the recommendation, a survey was done by an independent quality assurance team through questioner with the development and management teams. This evaluation was done in grader from 0 to 100. Average teams' feedback and actual project delivery statuses are takes into consideration.

Requirement management and requirement development average feedback results are shown in figure 4 and 5 respectively.

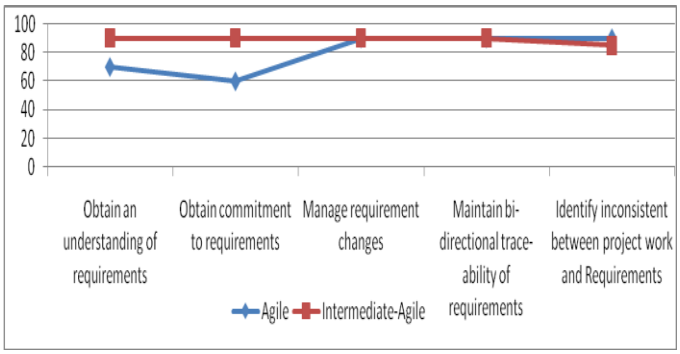


Figure 4: Requirement management process area experiments results

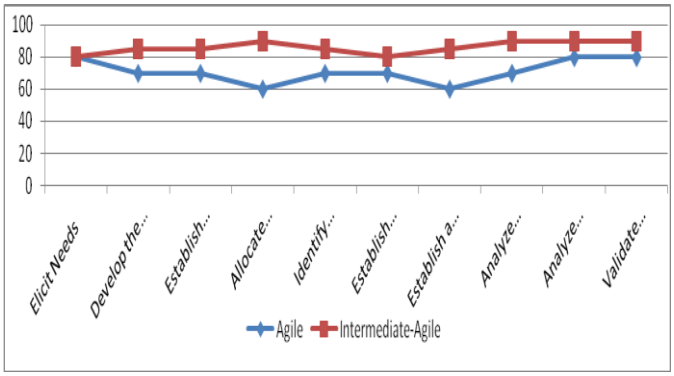


Figure 5: Requirement development process area experiments results

Example of these results description, regarding the goal "Obtain understanding requirements", it means that appropriate requirements' providers should be identified in order to avoid requirements overlapping or conflicts. This requires fixing requirements resources along with the project life (as much as possible). Regarding understanding of requirements, it really depends on the project structure, members and requirements providers' skills. But I-Agile is around 20% better in this point due to initial requirement phase, which enhance the requirements understanding and decrease the conflict between them. This enhancement is relative to project complexity.

5 Conclusion

Applying this paper recommendations in real projects affect the quality of the output and help in alignment with project schedule and budget. The experimental projects were small and medium size projects, applying paper recommendations in bigger projects size could present more positive or negative I-Agile model effect. The current I-Agile model care only about the requirements management and development, but also more deep studies in other project's phases and activities (other than requirements management and development) will lead to more complete I-Agile model, which could be a complete model used in all projects life cycle.

6 References

[1] Rodríguez, P., A. Zagüe, et al. "Some Findings Concerning Requirements in Agile Methodologies", Lecture Notes in Business Information Processing, 2009, Volume 32, Part 4, 171-184

[2] CMMI Official web site http://www.cmmi.de/cmmi_v1.2/browser.html?lang=en\lhs:null#hs:RDSG1

[3] Glazer, H. (2010). "AgileCMMI" from http://www.agilecmmi.com/

[4] Highsmith, J. (2001). "The Agile Manifesto" from http://agilemanifesto.org/

[5] Huo, M., J. Verner, et al. "Software Quality and Agile Methods", Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04)

[6] Sen, Z. and Y. Zheng. "The Relation of CMM and Software Lifecycle Model", ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing.

[7] Rosalva E. Gallardo-Valencia, Vivian Olivera, Susan Elliott Sim "Are Use Cases Beneficial for Developers Using Agile Requirements?",Fifth International Workshops on Comparative Evaluation in Requirements Engineering (CERE'07)

[8] Azida Zainol, Sa'ad Mansoor "Investigation into Requirements Management Practices in the Malaysian Software Industry", IEEE computer society,2008.

[9] Justin J.Y. Lin and Yung-Sung Lin "Collaborative Requirement Management System Developed for CMMI-Coherent Software Engineering", 2007 IEEE.

[10] Wikipedia web site http://en.wikipedia.org/wiki/Requirements_management

[11] http://www.agilecmmi.com/

[12] http://agilemanifesto.org/