Managing a Global Software Project under an Agile and Cloud Perspective

Giulio Concas¹, Katiuscia Mannaro¹ and Luisanna Cocco¹
¹Department of Electrics and Electronics Engineering, University of Cagliari, Piazza d'Armi
Cagliari, Italy

Abstract - Nowadays Lean-Kanban approach is perhaps the fastest growing Agile Methodology in software engineering. At the same time Cloud Computing (CC) is a technological phenomenon that is becoming more and more important in these last years. In our opinion Small and Medium Enterprises (SMEs) can increase their competitiveness by taking advantage of CC, and we think that it is very important to study and assess its impact on SMEs’ management processes. In this paper we proposed an effective tool to support strategic initiatives to the software development for the companies that develop software using agile methodologies and distributed resources. We used System Dynamics to model and simulate the software development: it allowed us to highlight in a very efficient way the interaction among several factors present in the software project.

Keywords: System Dynamics, Modeling, Simulation, Agile Processes, Global Software Development, Cloud System

1 Introduction

Nowadays, software engineering involves people collaborating to develop software and in this context many challenges, such as geographic, cultural, linguistic and temporal [4], [18], meet into Global Software Development (GSD). Some problems are related to the issues about the communication for information exchange, coordination of teams, and activities.

Normally, the distance and the lack of overlapping working hours create a negative impact on software projects, indeed problems in the knowledge transfer and, as a consequence, communications gaps or ambiguity on technical aspects must be resolved.

Cultural diversities may bring to an unequal distribution of work, lack of trust and fear, from which cost increases, poor skill management and reporting issues may arise. Linguistics and temporal diversities can instead lead to issues in knowledge transfer, communication and project visibility.

In our opinion, Cloud Computing (CC) allows us to deal with better all these problems.

CC is a delivery model for software, platforms and infrastructures. Cloud providers have got the possession of physical location, hardware, and system maintenance. Enterprise users access cloud services via the Internet from anywhere and at any time. Users usually pay a subscription fee, and can run a single instance of system on a robust infrastructure.

Indeed, Cloud services are delivered from a “multi-tenant” system; there is a single instance of software running, but many individual or enterprise customers use this system along with their own necessities.

In this paper we proposed a tool for managing an agile development environment on cloud platforms. This tool may support companies with distributed resources to take strategic decisions, no matter whether the choice involves outsourcing development or supplier networks. Software engineering involves people collaborating to develop better software. Therefore, we use collaboration tools all along the product life cycle to let us work together, stay together, and achieve results together.

This management tool is based on a model that we built by using an analysis of feedback loops among the components of the process, such as requirements, iterations, releases and so on, and through workflows and delays, in order to control their dynamics. We used System Dynamics to model and simulate how effective are Cloud-based software development environments for Global Software. We assumed a development process based on Scrum agile methodology and simulated the agile software development process on Cloud platform using a commercial tool available on the market: Vensim.

The proposed model helps managers to highlight all the factors that influence the software development in the companies with distributed resources. Indeed, in our opinion, our tool can be useful to improve all the activities linked to the software development. It allows us easily to highlight and focus all the elements that influence and compromise the success of a software project. Consequently, it allows us to discover, and then, to correct problems or conditions that could compromise the success of the project.
The remainder of the paper is organized as follows. Section 2 presents a brief description of some key software concepts of the two studied software development approaches and the Section 3 presents some related works. Finally Section 4 describes the details of the simulation model and Section 5 gives some final considerations of our research, and the recommendations for future works.

2 Optimizing the Software Development with Agile Methodologies and Cloud.

In this section we take a look at the considered software development approaches.

Scrum is presently the most used Agile Methodology (AM) [1], while the Lean-Kanban approach is perhaps the fastest growing AM. Scrum and Lean-Kanban have been proposed as two possible solutions to quickly respond to changing customer requirements, without compromising the quality of the code.

Specially in real-life software projects having up-front planning and budgeting, waterfall-like approaches are still very used. The Waterfall model was introduced by Royce in 1970. This software approach requires that all process phases (planning, design, development, testing and deployment) are performed in a sequential series of steps.

Each phase starts only when the previous one has ended. It is possible to step back to the previous phase, but it is not possible to go back in the process, for instance in order to accommodate a substantial change of requirements. This methodology requires defining a stable set of requirements only during the phase of requirements definition, and feedbacks to previous stages are not easily introduced.

Agile Methodologies, so named in 2001 in the Agile Manifesto [20], have been introduced in response to rigid and hard methodologies to follow. Among them, Scrum and Lean-Kanban are Agile process tools [8] based on incremental development. They both use pull scheduling and emphasize on delivering releasable software often.

The original term Scrum comes from a study by Takeuchi and Nonaka [19] that was published at 1986 in the Harvard Business Review. In 1993 Jeff Sutherland developed the Scrum process at Easel Corporation, by using their study and their analogy as the name of the process as a whole. Finally, Ken Schwaber [15] [16] formalized the process for the worldwide software industry in the first published paper on Scrum at OOPSLA 1995. Scrum [17] is a simple agile framework, adaptable also to contexts different from software development [12].

Adopting Scrum implies to use timeboxed iterations and to break the work into a list of smaller deliverables, ordered according to a priority given by the Product Owner. Changes to requirements are not accepted during the iteration, but are welcomed otherwise. Scrum projects are organized with the help of daily Scrum: 15 minutes update meetings, and monthly Sprints, or iterations, which are designed to keep the project flowing quickly.

Generally, at the end of every iteration the team releases working code, and a retrospective meeting is held also to look for ways to improve the process for the next iteration.

Lean software development is a translation of Lean manufacturing [6] to the software development domain. The Lean approach emphasizes on improving the value given to the customer, by eliminating the waste (Muda) and considering the whole project, avoiding local optimizations.

Kanban is a Japanese term that translated literally means visual (Kan) and card or board (ban). Adopting Kanban means to break the work into work items, to write their description on cards, and to put the cards on a Kanban board, so that the flow of work is made visible to all members of the team, and the Work in Process (WIP) limits are made explicit on the board. The Kanban board provides a high visibility to the software process, because it shows the assignment of work to the developers, it communicates priorities and highlights bottlenecks. One of the key goals of Lean-Kanban approach is to minimize WIP, so that only what is needed is developed, there is a constant flow of released work items to the customer, and developers focus only to deliver a few items at a time. So, the process is optimized and lead time can be reduced.

In a nutshell, Scrum and Lean-Kanban approaches are both agile processes aiming to quickly adapt the process by using feedbacks loops.

In Lean-Kanban the feedback loops are shorter, and the work does not flow through time-boxed iterations, but flows continuously and smoothly. Kanban is less prescriptive than Scrum and it is able to release anytime, while Scrum will release new features only at the end of the iterations. Moreover, in Scrum it is not possible to change the requirements in the middle of the sprint.

A common definition of Global Software Development is a software development process at geographically separated locations. For this reason, GSD involves communication for information exchange, coordination of teams, activities and artifacts so they contribute to the overall objective, and finally control of teams. Many challenges meet into GSD [4], [18] these are geographic, cultural, linguistic and temporal. The distance and the lack of overlapping working hours create a negative impact on software projects, create problems in the knowledge transfer, and as a consequence communications gaps or ambiguity on technical aspects may occur. Cultural diversities may bring to an unequal distribution of work, lack of trust and fear, from which cost increases, poor skill
management and reporting issues may arise. Linguistics and temporal diversities can instead lead to issues in knowledge transfer, communication and project visibility. The GSD can be facilitated using the Cloud.

CC is a delivery model for software, platforms and infrastructures. Cloud providers have got the possession of physical location, hardware, and system maintenance.

3 Related Work

Our model stemmed from two works about Global Software Development.

In [4], Hossain, Babar, Paik, and Verner discuss the use of Scrum practices in GSD projects, and identify key challenges, due to global project distribution that restricts the use of Scrum. In [18] instead, the authors present the challenges encountered in globally dispersed software projects and propose to exploit Cloud Computing characteristics and privileges both as a product and as a process to improve GSD.

In a more and more globalized world the relationship between culture and management of remote work is an avoidable issue to face. So, they exploit CC proposing both a product and a process to manage the many challenges in terms of culture, management, outsourcing, organization, coordination, collaboration, communication, development team, development process and tool.

In [1] a practical experience in the application of some agile software development practices, as Scrum model, to Azure application development is described. Azure Services Platform is an application platform on the cloud and it offers PaaS capabilities, which allow application to be built and consumed from both on-premise and on-demand environments. This paper starts from several questions about the interactions between cloud computing and agile software development and it attempts to discuss their potential advantages. In fact the authors show how setting up a development environment on the Azure platform helps enhance the agile practices.

Our work is based on a simulation technique used to study and analyze the software development. The used technique is known as System Dynamics.

The System Dynamics approach was introduced by Jay W. Forrester [7] of the Massachusetts Institute of Technology during the mid-1950s, and is suitable to analyze and model non-linear and complex systems containing dynamic variables that change over time.

System dynamics modeling has been used in similar research on software development process, where there are multiple and interacting software processes, time delays, and other non-linear effects such as communication level, amount of overtime and workload, schedule pressure, budget pressure, rate of requirement change, and so on. In the field of the Agile Methodologies, many system dynamics models were introduced. The main goals of these researches aim to better understand the agile process and to evaluate its effectiveness. Most of the performed research was made on Extreme Programming (XP), or generic AMs. Other processes such as Scrum, however, are almost absent.

For example, Chichacly in [2] investigated when AMs may work by using System Dynamics modeling, and comparing AMs with a traditional waterfall process. In [21] the author explored whether agile project management had a unique structure, or would fit within the generic conceptually formed system dynamic project management structures.

An analysis of factors that impacts on productivity during agile web development and maintenance phases was conducted by Xiaoying Kong et al. [10]. Another analysis published in [9] gives both theoretical insights into the dynamics of agile software development, and practical suggestions for managing these projects.

4 A Tool for the Global Agile Software Development on Cloud Environments

In this section, we describe a tool proposed to analyze and study the efficiency of a Cloud development environment used for Global Software Development. Since the collaboration among team members is an essential factor in GSD, this tool using Cloud resources is perfect to enable the facilitation, the automation and the control of all the development process.

The development methodology adopted in this environment set up using On-Demand resources is an Agile methodology known as Scrum methodology.

On the contrary of collocated software development, in GSD the distance among team entails difficulties in the coordination and the control due to problems which stem from many challenges in terms of culture, management, outsourcing, organization, coordination, collaboration, communication, development team, development process and tool among distant teams from each other.

We propose a tool that uses a simplified version of the Scrum approach in an On Demand development environment, in order to obtain a structure easy to understand and to modify during the whole life cycle of a software project.

According to SD modeling, and as reported in [11], our model is represented in terms of level variables, flow variables and auxiliary variables.

The tool proposed, shown in Fig.1, describes the development of a generic software project gone ahead by a small team of
developers. In order to simplify the model, all the phases of planning, design, coding, unit testing and similar have been merged into just one development phase, represented by the requirements development rate valve.

Figure 1. Tool for Cloud Software Development Environment.

The project software is modeled through a specific number of requirements, defined as a set of functionalities to be implemented. Therefore, the initial stock of requirements, which are represented by the level variable called “Original Work to Do”, evolves in a stock of developed requirements which are represented by the level variable called “Live”.

For the purpose of modeling a planning phase, which matches the real planning phases in the software development, we introduced in the tool some variables which represent the time spent in planning. They were modeled as delays in the software project development, which influence project outcomes and determine the system development speed.

On the contrary of traditional development environments, in Cloud development environments, the infrastructure is readily available, and system maintenance and system updates of the cloud server will be bear by the cloud providers and not by the developers.

The values of these variables will be linked to the set up of the Cloud development environment, in order to customize it as a function of their own needs.

The time and effort spent at the beginning of project development lifecycle are modeled by the following auxiliary variables: setting up infrastructure hardware and software licenses, deploying skilled resources to setup, manage and certify the software development and deployment infrastructure, building applications from multiple locations when teams geographically distributed are added.

Some of these variables just cited, were taken from the work of Dumbre et al. [1] and all are reported in Fig. 2.

Figure 2. Tool for Cloud Software Development Environment: Delay.

In addition to these variables, another variable for planning and creating the Backlog is taken into account. This variable is indicated as planning phase and it models the role of the Product Owner.

Indeed, Scrum prescribes roles, such as the role of the Product Owner. This role is given to a single person, who represents the customer's interest, prioritizes the requirements in the backlog, and answers to questions about requirements.

Moreover, Scrum prescribes a Sprint Planning Meeting, a Sprint Retrospective Meeting and Scrum of Scrums meeting, respectively, to plan the Sprint, to plan the iteration at the end of every sprint and to coordinate more teams which work at geographically separated locations.

As regards the modeling of the software development phase, as well known, the life cycle of the software project depends on the productivity of the developers and on the error made by them (see Fig. 1), but also by the uncertain customer requirements. Therefore, the fewer errors there are during the process, the sooner the project will be finished.

In according to [2], we modeled the auxiliary variable productivity: it represents the productivity of the developers. We take into account only these factors that in our opinion can be considered very relevant to the software development processes. However further factors can be easily introduced.

The factors taken into account are: the personnel experience, the personnel turnover, the communication complexity, the amount of overtime and workload, the schedule pressure, and the budget pressure (see Fig. 3).

Figure 3. Tool for Cloud Software Development Environment: Productivity.
So for example in Fig. 3 the *personnel experience* auxiliary variable contributes to the value of the *productivity* variable introducing a multiplicative factor that takes into account the knowledge of the current domain by developers. The *schedule pressure* auxiliary variable contributes to the value of the *productivity* variable introducing a multiplicative factor that takes into account the effect of the project falling behind the time schedule. Finally, for example the *communication complexity* auxiliary variable contributes to the value of the *productivity* variable by introducing a multiplicative factor that takes into account the effect present primarily in large project teams, where a large number of involved people increases the number of communication paths.

For each iteration, only a fraction of the requirements, in the level variable called “Selected Requirements”, is completed. This is because a fraction of the work is done incorrectly due to three types of error: effect of uncertain customer requirements, problem in the software design, bug introduced during the development.

Only the *bug introduced during the development error* passes through the rework discovery in Scrum valve. The two other errors can be discovered only at the end of the iteration.

Note that the requirements have the same size and weight and before to be developed are subdivided in different Sprint backlogs. Each backlog includes a random number of features, extracted from a Gaussian distribution. These Sprint backlogs are developed during short fixed-length iterations.

As requirements are implemented, they flow into the level variables “Integration Testing”, “System Testing” and “User Acceptance Testing” stock. If the tests are successfully passed, then the requirements are accepted and considered completed. Consequently, the accepted requirements flow into the level variable called “Production Environment”. Otherwise, a rework must be performed. This rework entails a delay due to the time needed for the correction. From “Production Environment” level variable the requirements flow into the “Live” level variable and the work is finished.

As we have already said, in our model, the time to finish the work “Original Work to Do” is affected by two main effects: delays and errors.

In Cloud system, Production and Testing environments are accessible anytime and anywhere. Any team member and user can work and build applications referring just to one location, with no need to coordinate multiple locations. In this way, significant time and effort will be saved, and users will use all their resources for creating value for their business.

Indeed, little time and efforts are spent to write verbose installation scripts or release notes, and for the setup of system testing, integration testing and user acceptance testing, with the aim to obtain a product released under rigorous test and validation. The deployment process is simplified, there is no need of any separate packaging efforts; to pass from development environment to testing environment, and from here to production environment does not require any additional step.

Prototypes and demos can be made accessible immediately to customers for eliciting feedback in a short time. The code can pass from one environment to another without writing deployment script to set up the application in the respective environments. All these activities have been modeled by two variables: *creating and managing different test environments* and *creating and managing production environment prototyping and demos* introduced when the requirements are moved from work done to test to a different testing environment, and then are deployed to production environment (see Fig. 4).

![Figura 4. Tool for Cloud Software Development Environment: Test Environments.](image)

All the variables described above represent time and effort spent in the development process.

## 5 Final Considerations and Future Work

In recent years, a new way to distribute and use the information and the communication technologies are heavily gaining ground at the expense of the traditional information and communication technologies. This new technology is the Cloud Computing.

This work analyzes and studies this new technology applied to the Software development process. So, we propose a model based on System Dynamics for highlighting the efficiency of the Cloud Platforms for Global Software Development.

We underline that the modeled software development process is based on agile methodologies. In particular, we applied a Scrum process, and hence an agile methodology able to manage better the software development with respect to the heavy and prescriptive traditional methodologies to develop as can be Waterfall process.
The realized model is a simple tool, this can be customized and used in order to follow the software development among its geographically distributed teams. Such a development environment allows to reduce the costs and the time with respect to an environment set up On-Premise, and hence a traditional environment. We developed a simplified model to describe all the significant factors that, in our opinion, enter during the life cycle of a software project. However, further factors can be easily introduced, and hence, the model can be easily customized to analyzing and studying real software project management. The modeled real development environment is very complex, and so, the model has been simplified. In addition, given the lack of experimental data, our goal is only to propose a tool to be used to help the software companies to plan and develop a software project. Moreover our study has been carried out under some limiting assumptions that could threaten its validity. The proposed model needs to be further elaborated and validated, for example by adding new variables or new relationships among factors. This work must be considered only a starting point. Indeed, given the lack of data available, here we do not show the results obtained simulating it. But, we would like to underline that very interesting results could be obtained to simulate it with real data from real software development experience. Therefore, the tool proposed will be the subject of our future work, that will include studies to empirically validate the model using data from GSD real projects and carried on using Cloud environments.

6 References


