Software code generator in Automotive field

Shahab Nadir
Technical university of Ilmenau
Ilmenau, Germany
sh.nadir@gmx.de

Prof. Detlef Streitferdt
Technical university of Ilmenau
Ilmenau, Germany
detlef.streitferdt@tu-ilmenau.de

***Abstract*— Rapid development of new technology has resulted changes in IT world affecting the way we work. Software use has increased in different fields of technology. In automation field, for example, the usage of Electronics Control Unit (ECU) has increased resulting in a different size of software. Due to persistent role of software in technology, the software cost and quality has become an important field. In automation field a number of tools are being developed which help to generate software codes. These tools can be used in software cost reduction because of reuse of software modules. In automatic code generation, the size estimates can hide the true effort of a program. In this paper we discuss the effect of using code generators in automation sector and how these tools have an effect on the cost of the software. We will compare the effort of code written manually and by using a code generator.**

Keywords- code generator, software engineering, automotive, code effort.

I. INTRODUCTION

In automation field, Original Equipment Manufacturers (OMEs) try to reduce the cost of their equipment. One important aspect that can reduce the overall cost is to reduce the cost of the software that is used in several applications. In quest of cost reduction, several OMEs decided to develop software platform which can be used in several variant of applications. These software is used to transfer data between different ECUs automobile by using data buses. One of the widely used buses in vehicle technology is Control Area Network (CAN). CAN needs a module (CAN stack) that makes the interface between the application and the bus. This module is used to read/ write messages on this bus. In automation field one vital platform that is extensively used for different applications is AUTOSAR (AUTomotive Open System ARchitecture). AUTOSAR is a multilayer software architect and provides the standardization of basic system functions and functional interfaces. Figure 1 show a general structure of Autosar. The fundamental idea of AUTOSAR is the reuse of software components and to master the emergent complexity of automotive electronic and software architectures [15]. It provides an interface between the application software components and other communication links e.g. CAN by using an interface layer 'Real Time Environment' (RTE). Different code generation tools have been developed for such layers to reduce the cost of software. These tools are used to generate the interface between the busses and the application. Using these tools a massive number of Lines of Code (LOC) can be generated with less amount of effort as compared to writing the same code manually. Some code generator tools are used to generate the CAN stack to transfer data between RTE and CAN bus, for example, CANgen/ Geny developed by Vector Informatik GmbH in Germany is an important tool which mostly used by OEMs [15]. In order to study the effect of the code generator tools, we have used two different code generator tools to generate the CAN stack for a specific ECU with the same environment and application. One with an easy configuration GUI (Graphic User Interface) form and other with a little complex GUI form which need more information to be configured and acquire the code. These two tools have been used by a number of engineers with the same document about the use of every tool, other engineers group are asked to implement the code manually. A second part of this experiment is done by the study of several software applications to advertise some CAN messages and signals information on a display. Some of these applications were written manually and some others by using a tool to generate the code. The programming language used is embedded C.

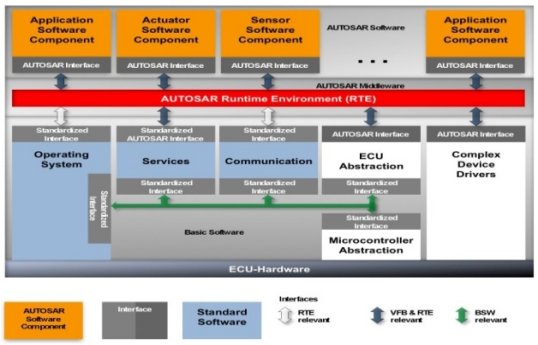

Figure 1 [15]: Autosar overview

II. METHOD AND STRATEGY

A. *ESTIMATING THE COST OF SOFTWARE*

There are several methods used to estimate the cost of the software. One of widely used method is COCOMO (COnstructive COst MOdel) developed by Boehm 1981 [1]. COCOMO estimation relies upon the size estimation and the environment information. Size estimation is product specific and is based on specifications and design plan while environment information is organization specific and includes type of the project, experience

978-1-4673-9795-7/15 $31.00 © 2015 IEEE
DOI 10.1109/CSCI.2015.186

of the staff etc., [2]. Size of the software is an important factor for the cost estimation in COCOMO method and this cost estimation is based on software metrics. Other important attentiveness factors in COCOMO method are complexity of the software, and the programming language used. With code generator tools the impact of these factors can change, for example, the LOC is not a big effort using these tools but here we should also take account of the cost and effort of the code generator tools in order to calculate the total cost of the software or software models.

B. Final Estimating the Effective Size of Auto-Generated Code in a Large Software Project

Barry Boehm in Software Cost Estimation with COCOMO II writes in [1] "Code generated with source code generators is handled by counting separate operator directives as source lines of code. It is admittedly difficult to count "directives" in a highly visual programming system. As this approach becomes better understood, we hope to provide more specific counting rules". Counting auto generated code is a tough task because using auto generated tools effort, productivity, and density estimates can be skewed [17][18]. In this paper our focus is on estimating the effort of the software in auto generated code and manually written code. The sample analysed in this paper deals with a newly developed ECU with over 65000 LOC. The development process selected is the V-Model. Some modules of the software are written manually and some others are generated with different code generator tools, one used tool was CANgen (code generator I), which is developed by Vector informatik in Germany. CANgen is professional tools including an easy used GUI to configure the CAN bus. Another is developed to be used for general bus configuration for internal usage in an automotive supplier company, this tool can be used to configure some different automotive busses. It has a general configuration GUI which needs some effort to configure the details of used bus. Because of internally use of the software we will call it in this paper as CANtool (code generator II). A number of programmers with same experience are asked to implement the applications.

To analyse the effect of code generator tools on the effort of software, two steps are performed on software modules:

- Studying the direct effect of code generator tools through comparing
- Analyzing the effect of code generator tools on different software modules

C. Studying the direct effect of code generator tools

In first step of the experiment, a requirement to receive five messages from the CAN bus in order to advertise the speedometer and tachometer in an instrument cluster with the speed unit, and send the calculated speed of vehicle on the Can bus. A group of twenty (20) programmers implemented the code to satisfy these requirements by using two code generator tools and another group of twenty (20) programmers analysed and implemented the requirements manually i.e., writing the code per hand. Table 1 shows the average effort, in terms of LOC, using the code generator I, code generator II and manually writing the code.

LOC	Code generator I	Code generator II	man ually
100	8	16	16
1000	10	20	50
5000	16	24	80

TABLE I: AVERAGE CODE EFFORT IN HOURS

A. Analyzing the effect of code generator tools with different modules

In second step of this experiment, a real software development for an instrument cluster is analysed. Some modules of this instrument cluster are implemented by using code generator I tool while some others are implemented manually. Table 2 shows the estimation results of the auto generated tools and manually written code. The complexity level depend on the Interfaces between the different layers and components in the Software. The complexity 1 means a simple transfer of signal from the bus to RTE layer with no interface among the different components, complexity 2 is used when the signal need some information from less than 3 other different components, and the complexity 3 by the need to deals with more than 3 components to get the needed value.

idx	Mod ule LOC	Generated code estimation [h] generator II	Handwritte n code estimation [h]	Complexity
1	800		42	2
2	1100		60	2
3	1400		120	3
4	1600		220	3
5	2000	20		2
6	400	8		1
7	1000	24		3
8	1500	28		3
	1000		200	2
10	950		120	2
11	800		80	3
12	1100		90	2
13	400		45	3
14	500		40	2
15	550		45	2
16	600		50	2
17	800		80	3
18	1200	16		2
19	1200	20		3
20	1300	16		2

Table 2: Estimated results of auto generated tool and manually writing code(1 = low, 2 = medium, 3 = high)

III. RESULT

The generator tools help to generate a large amount of code by the use of a configuration GUI. These tools allow the software developers to produce massive amounts of code using less effort. However, parametric estimation models that use the code size as a primary input (COCOMO II, SEER-SEM, etc.) have no active means to handle the auto generated and manually written code jointly. If only auto generated code size is considered as the solitary input, it increases the risk of underestimating the development effort. Therefore the estimation effort requires more inputs metrics than LOC size. The requirements analyses, design, implementation and testing the code are other vital metrics to estimate the effort.

In Automation field the development is performed at different CMMI (Capability Maturity Model Integration) levels depending on the equipment and the security level of the equipment. Development includes several phases where each phase is subdivided into activities. The default activities according to the V-model development process are requirements analysis, product (system) design, model design, model implementation, models integration, system test, test Plans, verification and validation (V & V), project office support, configuration management/quality assurance (CMIQA), and manuals. The percentages of effort associated with these activities [16] are given below in Table 3.

Phase	Effort
Requirement Analysis	10 %
System design	15 %
Model design	15 %
Code and unit test	20 %
Integration and integration-test	7 %
System test	33 %
	100 %

Table 1: Percentages of effort associated with different activities

The percentage effort can be separated as testing and non-testing effort. The final percentages are as follow: code implementation 20 %, requirement analysis and design 40%, and test phase 40%.

Total effort = RA (Requirement analysis) + C (Code) + I&T (Integration and Test)

Total effort = 40% RA + 20% Code + 40% Test

Therefore we can say the code effort to implement the system (and is dependent on the programming language used) is only 20% of the total effort. Code complexity, code quality and the size of code is also included in this 20% effort. By COCOMO LOC is an important factor which affects the price of software but software price also includes the effort of requirements analysis and system design, and testing effort. Since the test cases are designed depending on the code, total effort is modified according to equation given below:

Total effort = 60% LOC + 40% Test

In code generating tools when new requirements arrive or a new module development is required, the 20% effort of the code will be directly affected. Since the design of the new module or requirements is crucial which requires requirements analysis as well, therefore, requirements analysis and system design would be indirectly affected. The usability of the generators tools is shown in figure 2. The tool I is comparatively easier to use. Although it helped to decrease the total effort of the software but we should also take in account the cost of the generator tool. If we take the cost of the generator tool as an additional effort, commercially it will be negative and the overall cost will be increased.

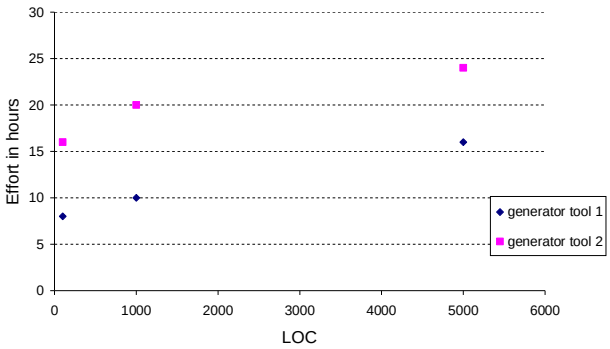

Figure 2: Usability of generator tools

Next we show the difference in effort between manually written code and auto generated code for different requirements in figure 3. The effort of LOC is depending on the complexity of the software requirement Table 2. In order to study the effect of generating tool on the effort of LOC, we measured some requirements with same complexity and determined the effort of 100 LOC of each requirement. In Figure 3 man can see that the code generator tools have more impotent for the requirements which need a huge LOC.

Code generator tools are useful for frequently used designs where the only requirement is to configure the tool with specific input parameters. Consequently there major benefit is for modules that are frequently used more than one time. Here auto generator tools can decrease the effort and cost of software development. The effort (E_d) can be calculated from the general software development equation:

$$E_d = C_s \,.\, C_{env} \,.\, S_{eff}$$

Where C_s Scaling constant, C_{env} Environment constant and S_{eff} Effective software source size, however the S_{eff} can be decreased by using a good tool generator in order to decrease the cost of the module.

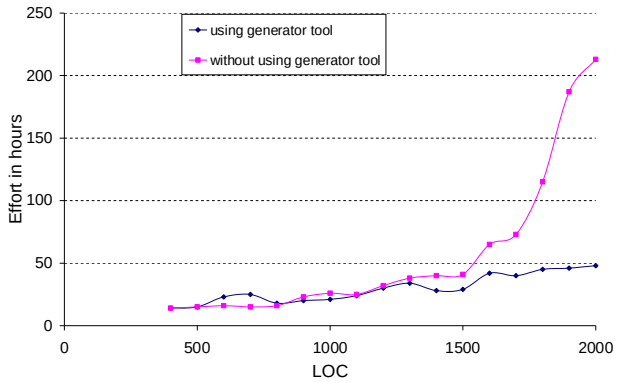

Figure 3: Difference in effort between manually writing the code and auto generated code

IV. CONCLUSION

In the last decades can man recognize that the use of software increased in Automation field. With this rising of software use, the complexity of software is increasing too. Develop the software with high quality has a huge effort and cost. Because of the several use of software pieces in different projects or different variants of project, the automatic software code generators are widely used in this field. The use of automatic code generation has proven to be a useful tool for the generation of software.

One of the development sector where the code generator are widely used, is to generate the basic software in ECUs. We used this approach to software development to replace and extend parts of an existing implementation of the common communication used by Autosar.

This development process provided several benefits over more traditional software development that are particularly important for safety critical software.

First, tightly coupled of the design and the implementation, so there is always an update for the documentation (design) with respect to the code. The implementation has no opportunity to drift away from the original design.

Second, using a tool to generate a different code types, and using a code generator with high usability can help to reduce the effort of development. Using generator tool could help to identify and correct large number of defects, before the C/C++ code is generated and run on the target hardware.

Third, updates and corrections to the design can be easily and quickly accomplished, the developer need only modify the diagram and regenerate the code. There is no need to hunt for affected regions in the code and manually update each one. Ultimately, these benefits can result in reduced development and testing time and a reduction in software development and maintenance costs.

An important consideration when using code generator is the usability of the tool. In this paper different tools are used to focus the importunacy of tool metrics. Although different metrices could be neede for different tools depending on the application of use.

ACKNOWLEDGMENT

Support for this work, by University of Ilmenau, was provided by Computer Science Department/ Software Architectures and Product Lines Group. And the authors would like to thanks in advance for all participated persons as subjects in this work.

REFERENCES

[1] Boehm, B., et. al., “Software Cost Estimation With COCOMO II”, *Prentice Hall PTR, Upper Saddle River, NJ*. 1999.

[2] Galorath Incorporated, “SEER-SEM User's Manual,” *Galorath Incorporated, El Segundo, CA*. 2000.

[3] Robert Ferguson, Dennis Goldenson, James McCurley, Robert W. Stoddard, David Zubrow, Debra Anderson , “Quantifying Uncertainty in Early Lifecycle Cost Estimation (QUELCE),” *SEI Identifier: CMU/SEI-2011-TR-026*, December 2011

[4] P. C. Pendharkar and J. A. Rodger, “A Probabilistic Model and a Belief Updating Procedure for Predicting Software

[5] Development Effort,” *IEEE Transactions on Software Engineering*, Vol. 31, No. 7, 2005, pp. 615-624. doi:10.1109/TSE.2005.75

[6] I. Stamelos, L. Angelis, M. Morisio, E. Sakellaris and G. L. Bleris, “Estimating the Development Cost of Custom Software,” *Information & Management*, Vol. 40, 2003, pp. 729-741. doi:10.1016/S0378-7206(02)00099-X

[7] L. Angelis, I. Stamelos and M. Morisio, “Building a Software Cost Estimation Model Based on Categorical Data,” *Proceedings of Seventh International Software Metrics Symposium, London*, UK, 2001, pp. 4-15.

[8] P. C. Pendharkar and J. A. Rodger, “The Relationship between Software Development Team Size and Software Development Cost,” *Communications of the ACM, forth-coming*, 2007.

[9] R. M. Stair and G. W. Reynolds, “Principles of Informa-tion Systems,” *Thomson-Course Technology, New York*, 2003.

[10] R. D. Banker and S. Slaughter, “The Moderating Effects of Structure on Volatility and Complexity in Software Enhancement,” *Information Systems Research*, Vol. 11, No. 3, 2000, pp. 219-240. doi:10.1287/isre.11.3.219.12209

[11] A. J. Albrecht, and J. E. Gaffney, "Software function, source lines of codes, and development effort prediction: a software science validation", *IEEE Trans Software Eng*. SE-9, 1983, pp.639-648.

[12] L. C. Briand, K. El Eman, F. Bomarius, “COBRA: A hybrid method for software cost estimation, benchmarking, and risk assessment”, *International conference on software engineering*, 1998, pp. 390-399.

[13] G. Cantone, A. Cimitile and U. De Carlini, “A comparison of models for software cost estimation and management of software projects”, *in Computer Systems: Performance and Simulation, Elisevier Science Publishers B.V.*, 1986.

[14] W. S. Donelson, “Project planning and control”, *Datamation*, June 1976, pp. 73-80.

[15] N. E. Fenton and S. L. Pfleeger, “Software Metrics: A Rigorous and Practical Approach,” *PWS Publishing*

Company, 1997.

[16] http://vector.com/vi_canbedded_j1939_functions_de.html?markierung=CANgen [Online]

[17] Central directive quality – "software development," *Robert Bosch GmbH Central Directive* [Online], May 2010.

[18] Ziegler, Stephen F., "Comparing Development Costs of C and Ada," *http://www.rational.com/products/whitepapers/337.jsp Rational Software, Cupertino*, CA. 1995.

[19] Softstar Systems, "A brief history of CoCoMoII," *http://www.softstarsystems.com, Softstar Systems, Amherst* [Online], *NH*. 2001