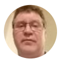
Estimation of Real-Time Software Code Size using COSMIC FSM

Conference Paper · March 2009

DOI: 10.1109/ISORC.2009.15 · Source: DBLP

Kenneth Lind
RISE Research Institutes of Sweden
16 PUBLICATIONS **130** CITATIONS
SEE PROFILE



Rogardt Heldal
Chalmers University of Technology
73 PUBLICATIONS **808** CITATIONS
SEE PROFILE

2009 IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing

Estimation of Real-Time Software Code Size using COSMIC FSM

Kenneth Lind
Saab Automobile AB
Trollhattan, Sweden
kenneth.h.lind@se.saab.com

Rogardt Heldal
Chalmers University of Technology
Gothenburg, Sweden
heldal@cs.chalmers.se

Abstract—**For distributed networks which will be mass produced, such as computer systems in modern vehicles, it is crucial to find cost efficient hardware. A distributed network in a vehicle consists of several ECUs (Electronic Control Unit). In this paper we consider the amount of memory needed for these ECUs. They should contain enough memory to survive several software generations, without inducing unnecessary cost of too much memory. Our earlier work shows that UML Component Diagrams can be used to collect enough information for estimating memory size using a Functional Size Measurement method. This paper replicates our earlier experiment with more software components of a different type. We compare the results from the two experiments.**

Keywords-Functional Size Measurement, COSMIC Function Points, UML components, software code size, system architecture

I. INTRODUCTION

The automotive industry integrates new features as software in a dedicated ECU (Electronic Control Unit) connected to a communications network. The downside of this integration method is that today's vehicles contain up to 100 ECUs. The system architectures of the future are expected to have fewer ECUs. It is vital that the ECUs have enough spare memory and spare processing capacity to last for the expected life of the system architecture. However, spare memory and processing capacity means additional cost, and the cost pressure in the automotive industry is increasing over time. Therefore, it is important to be able to estimate the amount of spare memory and spare processing capacity needed in a system.

In [1], we investigated FSM (Functional Size Measurement) as a method of estimating software code size based on requirement specifications. In this paper, we replicate the experiment in [1] with more software components, in order to expand the context of our results to a wider range of software component types. The work has been performed using available UML Component Diagrams within GM (General Motors), the producer of vehicles like Saab, Opel, Cadillac, Chevrolet, etc. Our goal is that our research shall contribute to a better balance between software code size and available memory. This is important for creating more cost efficient system architectures.

The paper is organized as follows: Next section provides background on UML components and FSM. Then we define the research problem, we describe how the study has been carried out, and the results are discussed. Finally, conclusions and suggestions for further work are presented.

II. BACKGROUND

A. UML Components

GM use Component Diagrams to show how the customer feature is divided into its smallest entities called "distributable components", and the interfaces between them. A distributable component is normally implemented in software, and executes on one ECU. A distributable component shall never be split up into more pieces, but can be used in several features. The experiment in this paper will use existing Component Diagrams of the type shown in figure 1.

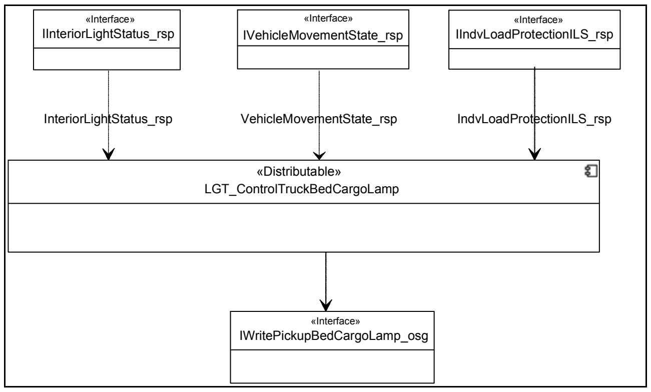

«Interface»
IInteriorLightStatus_rsp
«Interface»
IVehicleMovementState_rsp
«Interface»
IIndvLoadProtectionILS_rsp
InteriorLightStatus_rsp
VehicleMovementState_rsp
IndvLoadProtectionILS_rsp
«Distributable»
LGT_ControlTruckBedCargoLamp
«Interface»
IWritePickupBedCargoLamp_osg

Figure 1. Component Diagram of the Truck Bed Cargo Lamp feature.

In figure 1, we see that the distributable components are modeled as component stereotypes (denoted "Distributable" followed by the name of the distributable component). Signals (serial data signals, hardware I/O signals, etc) are modeled as interfaces and classes. Data transfer between distributable components or between a distributable component and an interface are modeled as flows. This notation deviates somewhat from standard UML 2.x [2]. For instance, flows are not usually part of Component Diagrams. The reason for GM to use flows is primarily to increase understandability among engineers not familiar with UML, which is important since the Component Diagrams are sent for review to stakeholders within the company to obtain feedback and approval.

The component diagram in figure 1 can be translated to standard UML 2.x, see figure 2. In this diagram required interfaces show what signals the component require from other components, and provided interfaces show what signals the component provide to other components. As we can see from the diagram, required interfaces are related to the component by a <<use>> dependency, and provided interfaces by a realization relationship. Signals are shown with the keyword <<signal>> in the interface.

A general description of software components is found in [3].

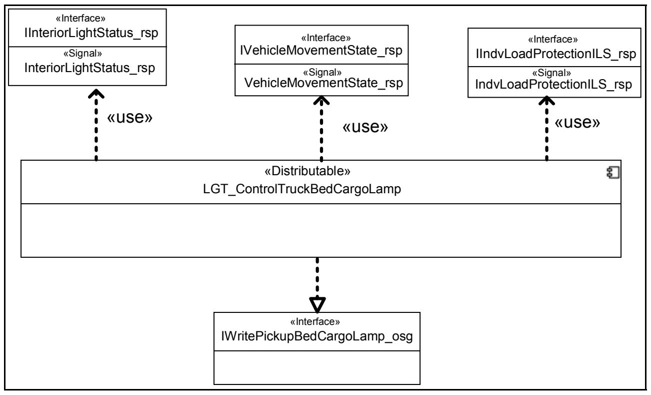


Figure 2. UML explicit representation of the Truck Bed Cargo Lamp feature.

B. *Functional Size Measurement*

There are several FSM methods available. A comprehensive literature survey covering several methods is found in [4]. In this experiment, COSMIC Function Points (CFP) is chosen because it is claimed to be suitable for real-time software, like automotive real-time systems [5], and it is a “second generation” method, complying to the ISO/IEC 14143-1:2007 standard for FSM methods.

The COSMIC Method defines a standardized measurement of software functional size, performed in three phases. The result of the method is a functional size measure expressed in CFP units.

The Measurement Strategy phase of the method, defines the purpose and scope of the measurement. The purpose of the measurement in this paper is to estimate the software code size based on the measured functional size. The scope of the measurement is defined based on the Component Diagram. This is described as part of the Experiment operation section of the paper.

The Mapping Phase of the method identifies the functional processes. A functional process is a component comprising a unique set of data movements (see figure 3 for types of data movements). This phase is based on the Component Diagram and the requirement specification, where the Functional User Requirements (FUR) can be identified. A FUR statement describes what the piece of software must do for the functional users (i e the senders or intended recipients of data to and from the required functionality).

The Measurement Phase is basically a calculation of the data movements involved in the required functionality. The types of data movements are defined in figure 3. There are four different data movement types:

- Entry/Exit sub-processes move data across the boundary to/from the functional process.
- Read/Write sub-processes move data from/to persistent storage.

Each data movement operates on a data group, and is equivalent to 1 CFP. A data group is a set of data attributes, where each included attribute describes a complementary aspect of the same object.

Persistent storage (see figure 3) enables a functional process to store a data group beyond the life of the functional process, or from which a functional process can retrieve a data group stored by another functional process.

By convention, the data manipulation of a functional process is assumed to be included in the data movements, and is not measured separately.

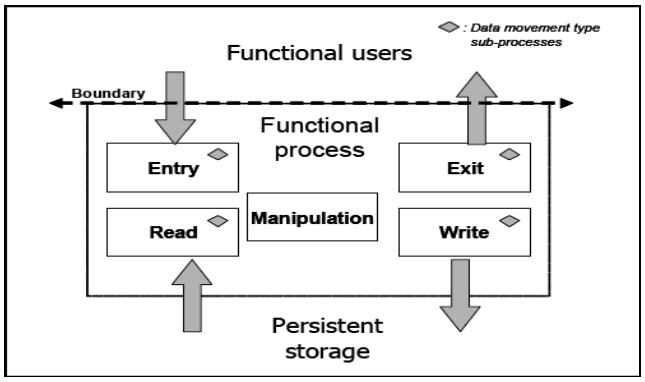


Figure 3. The Generic Software Model with data movement types [5].

III. EXPERIMENT DEFINITION AND OPERATION

The experiment carried out in this work is an extension to our work reported in [1]. In our previous work we investigated how to use COSMIC Function Points to estimate the software code size of software components developed by one development team within GM. We found significant correlation between CFP and actual software code size. Based on this correlation we developed a linear model capable of estimating software code size before the software is available.

However, we can anticipate that the experience of the development team might affect software code size. Other factors that might affect the software code size are software language, compiler, and development method. By investigating one team at the time, we minimize the variation due to these factors.

The intent of the experiment in this paper is to investigate how another development team using other methods to develop software components of a different type and for another ECU, influence the possibility to estimate the software code size. More specifically, can the linear model developed for the first team be used in both cases?

Requirement specifications and Component Diagrams are the two sources of input to the FSM method. The requirement specifications typically contain textual requirements on the performance, behavior, activation criteria, etc for a customer feature.

The Component Diagram is developed from the requirement specification and modeled in the Rhapsody tool [5], as part of the system architecture development activities within GM. So the Component Diagrams are already available for this experiment.

Our approach is to estimate software code size from CFP based on a linear model. A linear model is defined as y = k*x+b. The parameters k and b are calculated during the linear regression operation. Our statistical hypothesis needs to reflect whether the linear model is statistically significant or not. This is expressed as the following null hypothesis and alternative hypothesis:

H_0: All parameters in the linear model is equal to 0.

H_1: At least one parameter in the linear model is not equal to 0.

H_0 is rejected if the probability of the test statistic is less than 0.05.

A linear regression operation is evaluated based on checking the statistical assumptions for linear regression, how much variation in the data that is explained by the model (captured by R^2, the coefficient of determination), the probability by which the null hypothesis H_0 can be rejected (tested by an ANOVA test), and by estimating the data points not used for building the model.

The criteria for deciding if the model is good enough is that the residual between actual software code size and estimated software code size shall be within 10%.

Next, the operation of the experiment will be described. Around 300 software components are developed by GM for the Body Control Module (BCM) ECU. We want to study software components of similar type, so we concentrate on components of Comfort & Convenience type. They are characterized by a combination of event-based user inputs causing changes of one or several digital or analog output(s). There are roughly 100 software components of this type, and we chose 15 of them by random.

The COSMIC Method was applied to each of the software components in order to measure the size in CFP units. An example is shown below, where CFP is calculated for the LGT_ControlTruckBedCargoLamp component in figure 1.

1 Functional process:
- 5 Entry (Clock tic, System Power Mode, InteriorLightStatus, VehicleMovementState, IndvLoad-ProtectionILS)
- 1 Exit (WritePickupBedCargoLamp)
- 1 Read (PickupBedCargoLamp_Present)

CFP=7

We can see above that three of the Entry sub-processes InteriorLightStatus, VehicleMovementState, IndvLoad-ProtectionILS, and the Exit sub-process Write-PickupBedCargoLamp are obtained from the UML Component Diagram in figure 1. The other sub-processes are obtained from the textual requirements related to the component in the requirement specification, but could be incorporated in the Component Diagram.

After applying the COSMIC Method to each of the software components, the actual software code size in Number of Bytes of compiled code was obtained for the same software components. The data points are in the interval 4 CFP to 26 CFP, which is the anticipated size for the majority of the software components. So the data points are judged to be a representative subset of the type of components we want to measure.

IV. DATA ANALYSIS AND VALIDITY EVALUATION

As a first step we look at the scatter plot of the data, see figure 4.

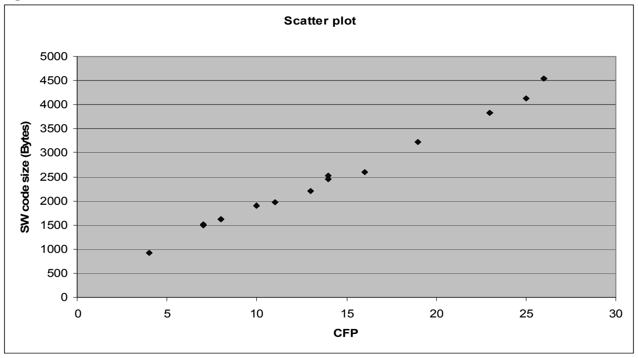


Figure 4. Scatter plot of experiment data, software code size (in Bytes) vs CFP.

As can be seen in figure 4, there seem to be a linear relationship between CFP and software code size. This is confirmed by calculating the correlation coefficients between CFP and software code size. So, the next step is to build a linear model based on linear regression, fitting a line that minimizes the sum of the quadratic distances to each data point. The general approach applied in this work is to choose a subset of the model build data, build a linear model based on linear regression, evaluate the resulting model, and compare it to other models. This is repeated for a number of possible subsets of the model build data.

The best evaluation results were obtained by using eight data points for building the model. The results in table 1 were given by using the model to estimate the data points. The values in the "% residual" column are calculated by dividing the difference between actual and estimated value with actual value. The data points used for building the model are highlighted.

As can be seen in table 1, the linear model is capable of estimating the software code size within 7% accuracy, including the data points not used in building the model.

The statistical data for the chosen model are R^2=0.996, F=1379.871, and probability of F statistic=0.000. Since the probability of the test statistic is well below 0.05, we can safely reject the null hypothesis.

The chosen model is defined by (1):

$$SWcodesize_{est} = 147.857 * CFP + 406.139. \quad (1)$$

As part of planning, performing and evaluating an experiment, it is important to consider possible threats to the validity of the results. [7] lists four types of validity threats: internal validity, external validity, construct validity, and conclusion validity.

Internal validity in our experiment is increased by a random selection of components to measure within the group of available components, and by minimizing the effect of methods and other factors in the experiment. On the other hand, it might be argued that internal validity is decreased by the limited number of data points. However, we believe that a random choice of 15 software components out of around 100 in total is enough.

External validity in our experiment is maximized by selecting a group of components that are representative of the type of functionality we want to measure. The components investigated in this work are of Comfort & Convenience type, characterized by a combination of event-based user inputs causing changes of one or several digital or analog output(s). This type of functionality is representative of at least 25-35% of the features in a typical vehicle today.

Construct validity in our experiment is affected by the fact that the COSMIC method is applied manually, with some room for interpretations. As a consequence, it cannot be guaranteed that another person would calculate exactly the same CFP measure. This could be improved by automating the calculation of CFP, and will be investigated in future work.

Conclusion validity in our experiment is affected by the power of the statistical test and if the assumptions for the test are violated. We have used an ANOVA test with the assumptions checked ok, and the resulting coefficients from linear regression are statistically significant.

Table 1. Evaluation of chosen model.

Component	Actual Software code size (bytes)	Est. Software code size (bytes)	% Resi - dual	C F P
Remote PRNDL Ill.	932	998	-7.0	4
Trunk Lamp	1492	1441	3.4	7
Truckbed Cargo Lamp	1504	1441	4.2	7
Panic Alarm	1612	1589	1.4	8
Front Zone Int. Lights	1614	1589	1.5	8
Rear Close Cargo Lamp	1908	1885	1.2	10
Power Sounder	1968	2033	-3.3	11
Ignition Switch Lamp	2202	2328	-5.7	13
Tonneau Release	2460	2476	-0.7	14
Dedicated DRL	2530	2476	2.1	14
IL Inadv. Load Protect.	2592	2772	-6.9	16
Horn	3218	3215	0.1	19
Interior Lights	3834	3807	0.7	23
Interior Dimming	4134	4103	0.8	25
Manual Liftgate	4530	4250	6.2	26

V. Interpretation of Results

As can be seen in table 1, the linear model is capable of estimating the software code size within 7% accuracy, including the data points not used in building the model. This is deemed as a good result. Based on this result we conclude that the model is valid within the interval where we have statistical data, i e 4 CFP to 26 CFP. This corresponds to the interval 998 Bytes to 4250 Bytes compiled code, which covers the majority of the software components of this kind.

The intent of this work was to continue our work reported in [1], by investigating another software development team developing components of a different type and how the results differ. The results from our previous work are included in table 2. The information in table 2 is obtained in the same way as in table 1.

Table 2. Evaluation of chosen model in [1].

Component	Actual Software code size (bytes)	Est. Software code size (bytes)	% Resi - dual	C F P
Speed Warning	388	402	-3.6	4
Engine Speed	556	550	1.2	6
Compass ind.	636	623	2.0	7
Vehicle Speed	740	697	5.8	8
Turn Signal ind.	922	918	0.4	11
Odometer	1122	1139	-1.5	14
Gear indication	1126	1139	-1.2	14
Driver Workload	1164	1139	2.1	14
Distance to Dest.	1242	1213	2.4	15
Outside Air Temp	1910	1950	-2.1	25
Gauges NP	2182	2318	-6.2	30
Display Dimming	4244	4529	-6.7	60

As can be seen in table 2, the linear model is able to estimate the software code size within 6.7% accuracy. So the two models perform with similar accuracy within their respective context.

The measurement data for both development teams are plotted in figure 5.

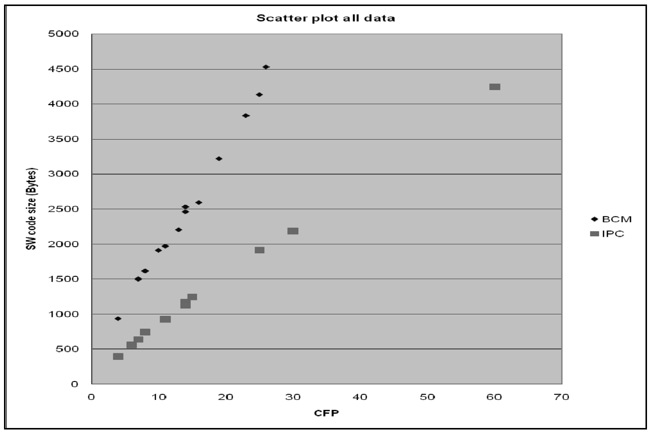


Figure 5. Scatter plot of experiment data for both development teams, software code size vs CFP.

As can be seen in figure 5, there is high correlation within the data for one development team. But, the correlation between the development teams is much lower. It is also clearly seen that the linear model developed for one development team will not perform well for the other development team. The reasons behind this difference will be investigated in future studies. For now we only indicate some possible reasons for this difference: methods and tools, team experience, software component type, software language and compiler, software infrastructure in the target ECU, requirement complexity not captured by COSMIC FSM, etc.

We have investigated how to estimate software code size based on Component Diagrams available before the actual software is available. Our idea is to estimate the code size of all software components contained in the ECU, add these up and add spare memory for expected feature growth. The result is the needed memory size.

This experiment concentrates on the part of the application software stored in ROM type memory. The size of RAM memory needed (for storage of variables and parameter values) is obtained fairly straightforward from the requirement specification, just by counting the variables and their size.

VI. Related work

The majority of published works on FSM methods deals with the problem of estimating development effort (man-hours) or development cost for a software program, based on requirement specifications, see [8] and [9]. The former paper [8] describes how COSMIC FP can be used in the telecommunications domain. It is shown that COSMIC FP is showing higher correlation to real values on telecommunication project sizes than IFPUG FPA. The latter paper [9] describes how IFPUG FPA can be used to estimate the total development cost, and to optimize the partitioning of hardware and software based on cost and reusability. The starting point for the proposed method is a system-level description in UML class diagrams and sequence diagrams.

Axelsson [10] perform work similar to our based on IFPUG FPA, but with little details on how to obtain the results and without empirical data. An example illustrates how IFPUG FPA can be used to evaluate different system architecture alternatives, based on development cost, product cost, and risk.

[11] applies Mk II FPA and COSMIC FP to a real-time software system. The results show that both methods can be used to measure real-time systems, and that COSMIC FP can be applied earlier in the development cycle than Mk II FPA.

[12] presents a Function Point like approach to measure the size of components using specifications written in UML. An interface complexity measure is combined with an interaction measure into a system-level size measure called Component Points. No empirical validation is reported.

[13] proposes a research area to handle the software complexity of data manipulations of a functional process. Resulting methods can be integrated into the COSMIC FP method to provide a more complete software size measure. This would complement the current COSMIC method as it only addresses the complexity of data movements.

VII. Conclusions and further work

This paper describes how UML Component Diagrams can be used to collect enough information for estimating software code size using an FSM method. We have used COSMIC Function Points as the FSM method, to calculate CFP on several UML components for which software is already developed. Significant correlation was found between the CFP values and the actual software code size. Based on this correlation we have designed a linear model capable of estimating the software code size of components within 7% accuracy. The linear model is valid for component sizes between 4 CFP:s and 26 CFP:s, a typical interval for this type of software components.

The software components in this paper are taken from the Body Control Module ECU, and are characterized by event-based user inputs causing changes of one or several digital or analog output(s) driving interior lamps, LEDs, electrical motors, etc. The software for the components has been developed by one software team using the same development methods and tools. The experiment is a natural continuation to our previous work, where we measured software components allocated to the Instrument Panel Cluster ECU. Those components mainly perform small calculations and display information of vehicle data, like vehicle speed, engine speed, etc. The found differences between software component types will be investigated in our future research. Further research will expand the context of our results, to include other software teams and other software component types.

References

[1] Lind, K, and Heldal, R, "Estimation of Real-Time System Software Size using Function Points", Proc. of the Nordic Workshop on Model Driven Engineering (NW-MoDE), 2008.

[2] OMG, Unified Modeling Language (UML), Superstructure, V2.1.2, Object Management Group, http://www.uml.org/.

[3] Szyperski, C, "Component Software: Beyond Object-Oriented Programming." 2nd ed. Addison-Wesley Professional, Boston 2002 ISBN 0-201-74572-0.

[4] Gencel, C, and Demirors, O, "Functional Size Measurement Revisited", ACM Trans. Softw. Eng. Methodol. 17, 3, Article 15 (June 2008).

[5] COSMIC, The Common Software Measurement International Consortium Functional Size Measurement Method, Version 3.0, Measurement Manual, 2007.

[6] Telelogic Rhapsody, http://modeling.telelogic.com/products/rhapsody/.

[7] Wohlin, C, Runeson, P, Höst, M, Ohlsson, M, C, Regnell, B, and Wesslén, A, "Experimentation in Software Engineering: An Introduction.", Kluwer Academic Publishers, 2000, ISBN 0-7923-8682-5.

[8] Afsharian, S, Giacomobono, M, and Inverardi, P, "A Framework for Software Project Estimation Based on COSMIC, DSM and Rework Characterization", Intl. Conf. of Software Engineering (ICSE'08), 2008.

[9] Fornaciari, W, Micheli, P, Salice, F, and Zampella, L, "A First Step Towards Hw/Sw Partitioning of UML Specifications", Proc. of the Design Automation and Test in Europe Conf. and Exhibition (DATE'03), 2003.

[10] Axelsson, J, "Cost Models with Explicit Uncertainties for Electronic Architecture Trade-off and Risk Analysis", Intl. Council on Systems Engineering (INCOSE), 2006.

[11] Gencel, C, Demirors, O, and Yuceer, E, "A Case Study on Using Functional Size Measurement Methods for Real Time Systems", Proc. of the 15th Intl Workshop on Software Measurement (IWSM), 2005.

[12] Wijayasiriwardhane, T, and Lai, R, "A Method for Measuring the Size of a Component-Based System Specification", IEEE The 8:th Intl. Conf. on Quality Software, 2008.

[13] Tran-Cao, D, Levesque, G, and Abran, A, "Measuring Software Functional Size: Towards an Effective Measurement of Complexity", Proc. of the Intl. Conf. on Software Maintenance (ICSM'02), 2002