PAPER Special Section on Software Reliability Engineering

CompSize: A Model-Based and Automated Approach to Size Estimation of Embedded Software Components

Kenneth LIND[†a)], *Student Member and* **Rogardt HELDAL**[††], *Nonmember*

SUMMARY Accurate estimation of Software Code Size is important for developing cost-efficient embedded systems. The Code Size affects the amount of system resources needed, like ROM and RAM memory, and processing capacity. In our previous work, we have estimated the Code Size based on CFP (COSMIC Function Points) within 15% accuracy, with the purpose of deciding how much ROM memory to fit into products with high cost pressure. Our manual CFP measurement process would require 2.5 man years to estimate the ROM size required in a typical car. In this paper, we want to investigate how the manual effort involved in estimation of Code Size can be minimized. We define a UML Profile capturing all information needed for estimation of Code Size, and develop a tool for automated estimation of Code Size based on CFP. A case study will show how UML models save manual effort in a realistic case.

key words: *UML Profile, UML components, software components, functional size measurement, code size estimation*

1. Introduction

Early and accurate estimation of Software Code Size is important for developing cost-efficient embedded systems. For this type of system the software and hardware must be developed in parallel due to time-to-market constraints. Examples are cars, cell phones, washing machines, etc. The Code Size affects the amount of system resources needed, like ROM and RAM memory, and processing capacity. Systems containing too much memory or processing capacity are more expensive than they need to be. Systems containing too little memory or processing capacity may need a redesign after only a part of its expected lifetime.

In our previous work, we have estimated the Code Size based on CFP (COSMIC Function Points) within 15% accuracy [23]–[28]. Our results were obtained using software implementations developed by the automotive companies Saab and GM (General Motors). The accuracy of the estimated values is important because the purpose was to decide how much ROM memory to fit into ECUs (Electronic Control Unit, an embedded computer) in products with high cost pressure. Our manual CFP measurement process used UML components and textual information from requirement specifications, which would require up to 2.5 man years of effort to obtain the CFP value for the application software embedded in a typical Saab car.

Manuscript received December 12, 2011.
Manuscript revised April 20, 2012.

[†]The author is with Viktoria Institute AB, Gothenburg, Sweden.
[††]The author is with Chalmers University of Technology, Gothenburg, Sweden.
a) E-mail: kenneth.lind@viktoria.se

DOI: 10.1587/transinf.E95.D.2183

In this paper, we want to investigate how the manual effort involved in estimation of Code Size can be minimized. The UML components provide some (but not all) of the information needed for estimation of Code Size. Therefore, we define a UML Profile capturing all information needed for estimation of Code Size [29], and develop a tool for automated estimation of Code Size based on CFP [30]. Besides the increased efficiency obtained by our model-based and automated estimation approach, we expect to increase repeatability and consistency in the estimation process compared to a manual approach.

In order to investigate if our approach solves our problem, we formulated the following research questions;
RQ1: "How can UML support in modeling all information needed for automated estimation of Software Code Size?"
RQ2: "How much manual effort can be saved by modeling all information needed for automated estimation of Software Code Size?"

We conduct a case study using requirement specifications and software implementations from the automotive industry to answer the research questions.

The contribution of this paper is a complete description of our model-based and automated estimation approach, and an explanation about how the UML Profile [29] and the CompSize tool [30] fit together. In addition, we aim to provide enough details to enable other researchers and practitioners to apply our approach.

The rest of this paper is organized as follows: The next section provides background information about the COSMIC method. Section 3 defines the UML Profile, and Sect. 4 presents the tool. Section 5 describes the case study, and Sect. 6 evaluates threats to validity. Sections 7 and 8 contain related work, and conclusions.

2. Background

This section presents enough information about Functional Size Measurement and the COSMIC method to understand the rest of the paper.

Functional Size is defined as "size of the software derived by quantifying the Functional User Requirements" [1]. FUR (Functional User Requirement) describes what the software is expected to do for its users. Examples are data transfer, data transformation, data storage, and data retrieval. Functional Size is independent of software language and development methods.

There are several FSM methods available. The original

method was described by Albrecht 1979 [2]–[4]. A comprehensive literature survey covering several methods is found in [9]. Some of them are IFPUG FPA (Function Point Analysis) [12], and COSMIC Function Points (CFP) [1], [19] to name a few. The typical usage of FSM is development cost estimation and project planning. In our work, CFP is chosen because it is known to be suitable for real-time software, like automotive systems [1], and it is a "second generation" method, complying with the ISO/IEC 14143-1:2007 standard for FSM methods [13]–[18].

The COSMIC Method defines a standardized measure of software Functional Size expressed in CFP units. The measurement is carried out in three phases; the strategy phase defines the purpose of the measurement and scope of the software to be measured, the mapping phase maps the FUR of the software to be measured onto functional processes in the software component of the COSMIC Generic Software Model (shown in Fig. 1), and the measurement phase counts the data movements contained in each functional process. By defining the purpose of the measurement and scope of the software to be measured during the strategy phase, we identify the level of decomposition and level of granularity of the software to be measured. The level of decomposition points out a particular level in a software – component – sub-component hierarchy. The level of granularity concerns the amount of details defined about the FUR. Both aspects are important when comparing different CFP values to each-other.

As can be seen in Fig. 1, there are four different data movement types. Entry types move data across the boundary and into the functional process. Exit types move data across the boundary to a user. Read types move data from persistent storage to the functional process. Write types move data from the functional process to persistent storage. Persistent storage (Storage Hardware in Fig. 1) enables a functional process to store data from one execution cycle to another. Each data movement is equivalent to 1 CFP, and operates on a common set of attributes.

In our previous work [27], we have identified factors to use for categorization of software in our domain. The categorization is important to increase the estimation accuracy, by using historical data from implementations of similar software. This way we can capture algorithmic complexity and manipulation of large amounts of data, although COSMIC cannot measure this directly.

To summarize this section, we conclude that the main concepts we need to consider in COSMIC are the Generic Software Model (containing users, boundary, functional processes, and data movement types), the level of decomposition, and the level of granularity. In addition, categorization of software is important for estimation of Code Size. How these concepts can be modeled in UML will be described in the next section.

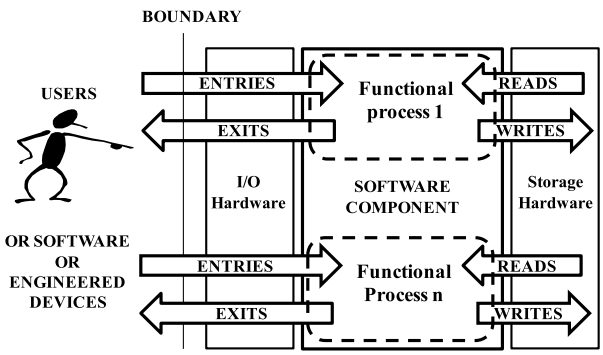

Fig. 1 The generic software model of COSMIC.

3. A UML Profile for Code Size Estimation Based on COSMIC

Our goal is to define how to model the COSMIC Generic Software Model using UML. We have chosen UML because it is commonly used for system architecture and software development, and it was already in use at Saab and GM. UML components [32], [37] have a natural boundary between the software and its users, in a similar way as in COSMIC. If we view the UML component as the COSMIC Generic Software Model (see Fig. 1), then we can view the Entry data movements as operations in required interfaces and Exit data movements as operations in provided interfaces.

The UML components do not capture the Read and Write data movements. Our idea is to capture these data movements by a UML class representing this information within the components. Each attribute of this class represents data that can be read, written, or both to/from memory. To achieve this we extend the Property in the meta-model [33] with the stereotype CompSizeProperty, see Fig. 2. We give the stereotype CompSizeProperty the attribute "direction" to model Read and Write data movements. The value "in" represents Read data movements and the value "out" represents Write data movements. The value "inout" represents a combined Read and Write data movement, which will be counted as 2 CFP.

Now we can represent all the data movements of COSMIC using UML components and our extended classes containing attributes with direction. UML components are often used to model complex systems by decomposing a larger software system into smaller parts. In this case we only need to extend the components with a class representing the Read and Write data movements to obtain software models which can be measured by COSMIC.

We can represent the COSMIC Generic Software Model using UML models, and from the models we can obtain CFP values. But how many bytes will 1 CFP represent?

Fig. 2 The CompSizeProperty stereotype.

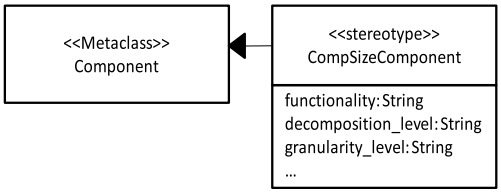

Fig. 3 The CompSizeComponent stereotype.

This might depend on several factors such as the decomposition level of the component, compilers used, type of functionality, development methods & tools, etc. These factors can be used to categorize the components into groups containing components of similar type. Here we will consider two key factors in some detail: level of decomposition and type of functionality.

In domains where components are used at different levels of decomposition, it has to be clearly marked for each component which level of decomposition the component belongs to. For example, components that describe the top level architecture can contain several other components and will therefore in most cases correspond to more code, compared to components on a lower level only containing classes. In the case study we will present later in this paper, all the components are specified at the same level of decomposition.

Another factor which might be important is the type of functionality. This factor is of particular importance if the different types of functionality correspond to different byte sizes. This factor gives the possibility to take into account the algorithmic complexity of the components. For the automotive domain we have shown that categorizing the components into groups of similar components is one of the key factors for our good estimation results [27], [28].

To be able to model the categorization information, we extend the Metaclass component with the stereotype CompSizeComponent (see Fig. 3). The stereotype has several attributes to be able to assign values to different factors. Exactly which attributes the stereotype should have may differ between domains, but some factors are probably general such as decomposition_level and functionality. We have included granularity_level which is explained in Sect. 2. Other factors necessary for categorization of components within GM and Saab are identified in [27].

Table 1 summarizes the complete mapping between the main COSMIC concepts and corresponding UML concepts.

With the UML models described so far, we can capture all information needed for accurate estimation of implemented Code Size in bytes. This bytes value corresponds to the amount of ROM-type memory needed to store the code implementation of the component.

The UML Profile defined in this section will be evaluated in a case study later in this paper. The case study will show a concrete example on how to use the Profile (see Fig. 10). The mapping rules summarized in Table 1 was implemented in a tool, which will be described in the next section.

Table 1 Mapping rules between COSMIC concepts and UML profile.

COSMIC concept	UML concept
Functional process	The functional requirements contained in the component. Must reside completely within one component.
User	Surrounding components.
Boundary	Component boundary.
Level of granularity	Part of categorization.
Level of decomposition	Part of categorization.
Entry data movement	Operation in required interface.
Exit data movement	Operation in provided interface.
Read data movement	CompSizeProperty with direction=in.
Write data movement	CompSizeProperty with
Read/Write data	CompSizeProperty with

4. The CompSize Tool

We have developed a tool based on our results from 3 years of research. The tool was implemented in Java JDK 1.6 using Eclipse IDE for Java Developers resulting in around 1.7 Mbytes of code, and required 6 man months of effort.

The main functionalities of the tool are; to import information modeled using the UML Profile described in the previous section, to store component data needed for CFP measurement and Code Size estimation, to calculate estimated Code Size using linear regression, and to present estimation results.

The main window of the tool is shown in Fig. 4. All the information about the components and categorization can be found in the Components Display tab in the upper section of the window.

Here you find the name of the component, the number of Entry, Exit, Read, Write data movements, the CFP value, estimated and real Code Size in bytes, etc. Historical data containing real Code Size for similar components are used to estimate the Code Size of new components. The DFP value in column 6 is defined in [28] as a measure that only counts the Entry and Exit data movements. The tool also supports calculation of DataSize (RAM memory size) and estimation of development effort. For that purpose it can store estimated and real DataSize, as well as estimated and real effort. In Fig. 4 you see the component named Truck Bed Cargo Lamp as highlighted. It has the values Entry=5, Exit=1, Read=1, Write=0, which amounts to CFP=7, and its Real Code Size=1504 bytes.

In Fig. 4 you also see the categorization factor values for the component. Its categorization factor values are Team=A, Functionality=Comf & Conv, etc.

Estimations are based on components with the same set of factor values. From the set of components, the CFP value and the Code Size are selected and a linear model is created using linear regression. In the "Scatter Plot" tab of the upper section of the window in Fig. 4 these values and the linear model is presented when the user is estimating a component, see Fig. 5. The scatter plot has CFP on the X axis, and Code

Component(s) Display | Scatter Plot

Component Name	Entry	Exit	Read	Write	DFP	CFP	Est. CodeSize	Real CodeSize	Est. DataSize
Idle Boost	7	1	17	0	8	25	0	1002	0
Dear Defrost	16	5	11	0	21	32	0	1772	0
Ignition Switch Lamp	11	1	1	0	12	13	0	2202	0
Tonneau release	11	1	2	0	12	14	0	2460	0
Dedicated DRL	6	3	5	0	9	14	0	2530	0
Manual Liftgate	11	4	11	0	15	26	0	4530	0
Panic Alarm	5	2	1	0	7	8	1570	1612	0
Power Sounder	7	3	1	0	10	11	0	1968	0
Remote PRNDL Illumination	2	2	0	0	4	4	0	932	0
Interior Dimming	19	6	0	0	25	25	4171	4134	0
ILS Inadvertant Load Protection	9	4	3	0	13	16	0	2592	0
Interior Lights	15	4	4	0	19	23	3865	3834	0
Truck Bed Cargo Lamp	5	1	1	0	6	7	0	1504	0
Trunk Lamp	4	2	1	0	6	7	0	1492	0
Rear Closure Cargo Lamp	7	2	1	0	9	10	0	1908	0
Front Zone Interior Lights	5	2	1	0	7	8	0	1614	0
Horn	10	1	8	0	11	19	0	3218	0

Component Name	Factors	
Truck Bed Cargo Lamp		
	Team	A
	Functionality	C&C
	Constraints	Frame
	Method&tool	Rhapsody

Fig. 4 Components display tab.

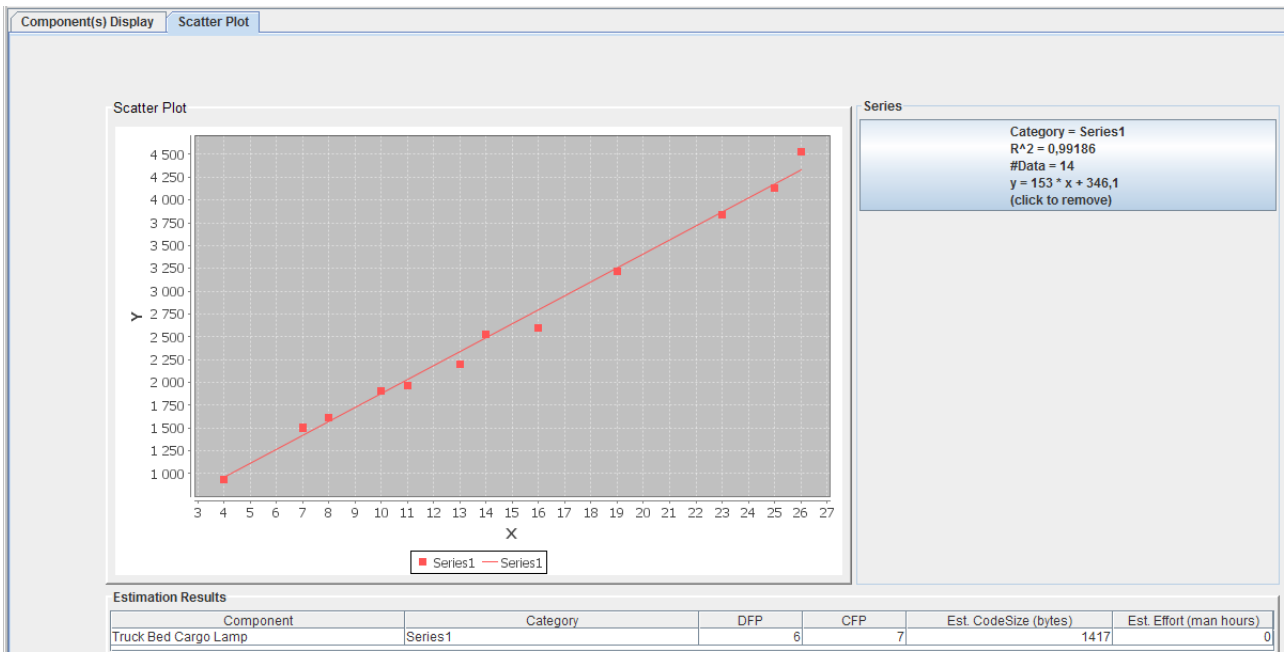

Fig. 5 Scatter plot tab.

Size in bytes on the Y axis. The squares in the plot show historical data for component implementations, and the line shows the linear model calculated using linear regression on the historical data. To the right in the Scatter Plot Tab, the category name (Category=Series 1 in this case) of the historical data is shown. As many components as possible should be filled with historical data as more values will result in a better linear model. The absolute minimum is 2 valid components in a set for estimation. This would most likely result in estimation with an unsatisfying margin of error, so to guide the user the estimation results contains information about the R^2 value (R^2=0.99186) of the linear regression and the number of valid components used (#Data=14). With experience the engineer should be able to tell by these values if the margin of error of the estimation is within acceptable borders for the current task. The equation of the linear model is also shown (y=153×x+346.1). In the lower part of the Scatter Plot Tab we see that the Truck Bed Cargo Lamp component has been estimated, that Category=Series 1 has been used for the estimation, that CFP=7, and that the estimated Code Size=1417 bytes. If the user is satisfied with the result it can be copied to the component by using the button in the bottom of the Scatter Plot Tab.

The Components Display Tab and the Scatter Plot Tab

are the main windows in the tool. Other windows in the tool are designed to guide the user how to perform a certain task. Examples are the Factors Manager Tab (where categorization factors can be added or removed), the Estimation Setup Tab (where estimations are initialized by selecting a component in the "Component Display" and then the Category and type of estimation (Code Size or Effort) can be selected), and the Add Category Tab (where categories can be defined as a list of factors and corresponding factor values).

The tool was validated during development to make sure that the different parts work as expected. Demo workshops were conducted to get feedback about the user interface from potential users of the tool. In the next section we will describe a case study we conducted to evaluate the tool.

5. Case Study

A case study consisting of several parts was defined, in order to evaluate our model-based and automated approach and to answer the research questions defined in Sect. 1. The case study was conducted at Saab using requirement specifications and software implementations developed by Saab and GM.

5.1 Definition and Planning

Saab and GM use UML Component Diagrams to show how the customer feature is divided into its smallest entities called "distributable components", and the interfaces between them. The distributable components are defined at a fixed level of decomposition. A distributable component must never be split up into more components, but can be used by several features. The UML Component Diagram is modeled in the Rhapsody tool [11], as part of the system architecture development activities within Saab and GM. This is described further in [5].

The case study will use existing Component Diagrams of the type shown in Fig. 6. In this diagram, we see that the distributable components are modeled as component stereotypes denoted "Distributable" followed by the name of the component. As we can see from the diagram, the Truck Bed Cargo Lamp component has three required interfaces and one provided interface. The Truck Bed Cargo Lamp component is taken from a real Component Diagram and will be used as an example throughout this paper.

The Component Diagrams do not contain all the information we need to measure the Functional Size. We also need the requirement specifications related to the components. In the requirement specification we find in textual form the information needed such as: calibration parameters (used for tuning of a general software component to a certain type of product), persistent storage of variables in RAM-type memory, etc. The textual requirements for the Truck Bed Cargo Lamp component in Fig. 6 are shown in Fig. 7.

Next we describe the process for estimating the implemented size of software components. The main activities

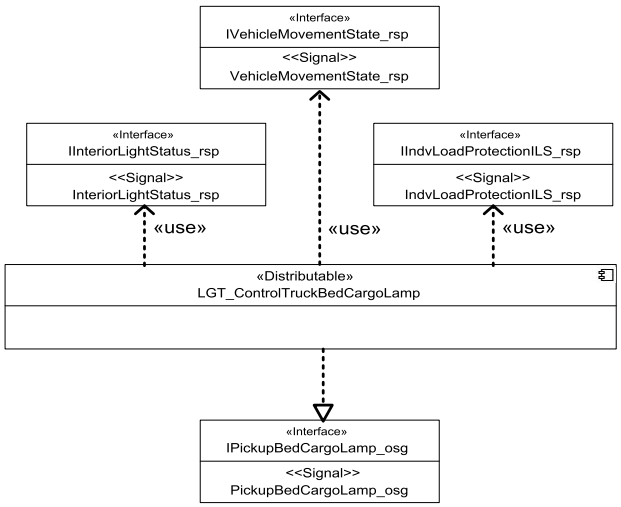

Fig. 6 Component diagram of the truck bed cargo lamp component.

The feature **Shall** be enabled when the calibration CARGO LAMP PRESENT is set true. <END>
If the vehicle power mode is "OFF", and the cargo lights are illuminated, the SYSTEM **Shall** keep the cargo lamps active as long as Inadvertent Load Control power is active. <END>

CUSTOMER "ACTION"	CUSTOMER PERCEIVABLE "OUTPUT"	MAXIMUM LATENCY "ACTION" to "OUTPUT"
INTERIOR ILLUMINATION Lamps Switch On and Vehicle Parked.	***Cargo Lamp Illuminates***	**100 ms**

Fig. 7 Extract from the requirement specification for the truck bed cargo lamp component.

(grey boxes) and artifacts (white boxes) involved in the process for estimation of software component size are shown in Fig. 8. The first activity is the definition of the functional requirements from a user perspective and the non-functional requirements, resulting in a textual specification. This is typically performed by an expert in the particular functional domain, e.g. a door locking expert rather than a software engineer. The textual specification is used by the architect to decompose the functional requirements into distributable components, which are modeled in UML Component Diagrams. The Component Diagrams are used for software design and implementation, as well as for serial data communication definition and implementation. These activities are left out in this description, since they are not important in this work. Instead, we will continue describing the activities performed by the measurement engineer, shown at the bottom of Fig. 8.

The Component Diagram is used to identify the interfaces, and the textual specification is used to identify the calibration parameters and the information needed for categorization of the distributable component. The interfaces, parameters, and categorization constitute the information needed by the COSMIC method. The mapping between the information and COSMIC is illustrated in [28]. The result from the COSMIC method is a CFP measure. Historical data containing CFP and implemented Code Size in bytes

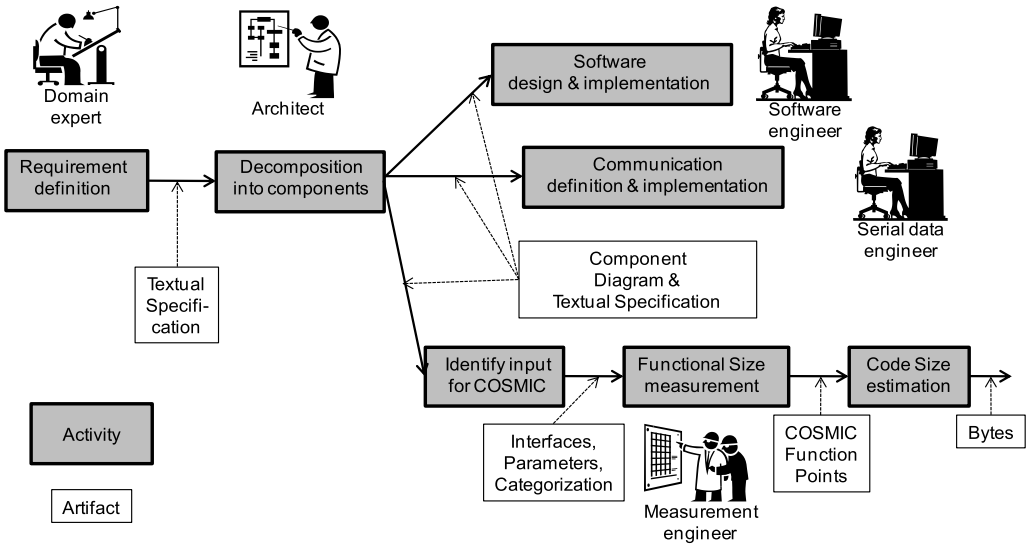

Fig. 8 Main activities and artifacts for estimation of software component size.

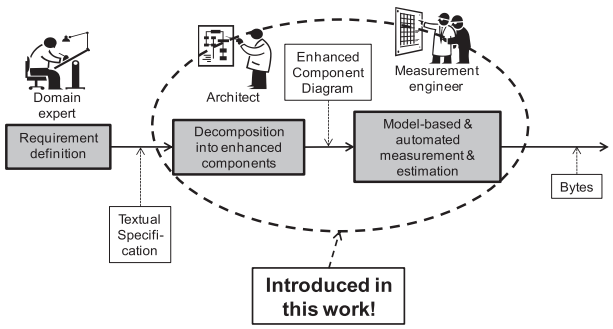

Fig. 9 Main activities and artifacts for the proposed model-based and automated approach to estimation of software component size.

for similar distributable components are used to convert the CFP value into bytes, and thereby estimate the implemented Code Size.

In our previous work, we have estimated 46 distributable components manually according to this procedure. The first author of this paper acted as measurement engineer, and spent 2-4 hours per distributable component for the activities "Identify input for COSMIC", "Functional Size measurement", and "Code Size estimation". To put this in a practical perspective, we estimate that a typical Saab car contains around 1200 distributable components. This means that it would take up to 4800 man hours (roughly 2.5 man years) to estimate the complete application Code Size of a car. Hence, manual estimation of Code Size is not feasible in this context. Instead we propose a model-based and automated approach, as described in Fig. 9.

In our proposed approach, the architect adds information about calibration parameters and the information needed for categorization of the distributable component into "Enhanced Component Diagrams". The Enhanced Component Diagrams was defined as a UML Profile in Sect. 3 of this paper. The Enhanced Component Diagrams are exported into an XML file containing all the information needed for COSMIC. Hence, the activity "Identify input for COSMIC" that was performed manually by reading the textual specification before, is performed by the architect who is already familiar with the requirements. The anticipated saving in manual effort is that the measurement engineer does not have to read the textual specification. The next step in Fig. 9 is that the XML file is imported to the CompSize tool that automates the "Functional Size measurement" and "Code Size estimation" activities in Fig. 8. The CompSize tool was described in Sect. 4 of this paper.

The case study was conducted in two steps. The first step of the case study concerns estimation of distributable components with given Component Diagrams. The manual measurements and estimations obtained in our previous work [28] according to Fig. 8 are replicated using the UML Profile and the CompSize tool according to Fig. 9. The purpose is to evaluate the UML Profile and XML import to the tool, and hence answer RQ1.

The second step of the case study concerns estimation of distributable components with unknown Component Diagrams. The manual measurements and estimations obtained in our previous work [25] according to Fig. 8 are replicated according to the complete process described in Fig. 9. Relative effort data are compared to absolute effort data obtained from interviews with architects. The purpose is to answer RQ2.

5.2 Operation and Data Analysis

The case study was defined, planned, supervised, and analyzed by the authors of this paper, but it was conducted by two Master students with no prior experience from the COSMIC method and with limited knowledge about the automotive domain. The reason that we let students conduct the case study instead of architects is that students are equally inexperienced with each phase of the estimation process as

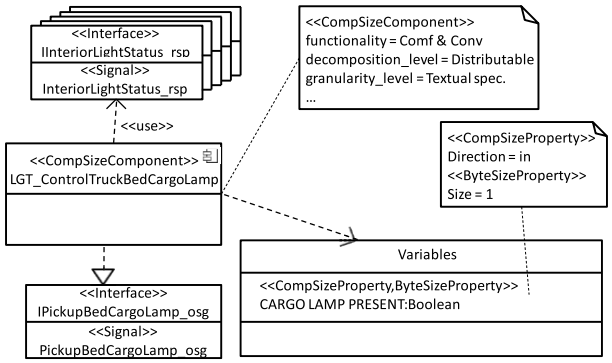

Fig. 10 Mapping of a distributable component onto the UML profile.

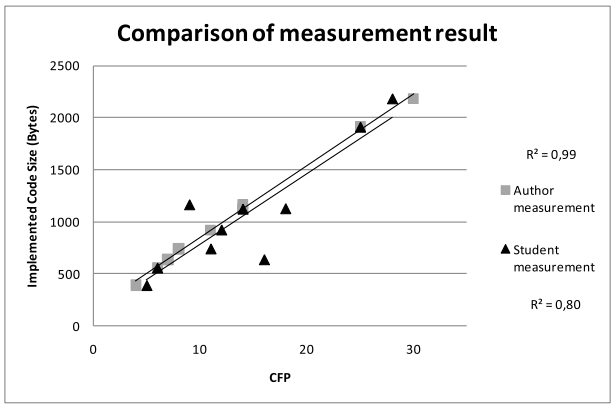

Fig. 11 Scatter plot showing CFP values measured by the first author and the students for the same components.

well as with different components. An architect on the other hand, is familiar with the Component Diagrams and can model that faster than the rest of the UML Profile. Therefore we can obtain effort measures that are less biased from the students. The students were given lectures about the COSMIC method, the format and structure of the textual specifications, and the UML Profile. The CompSize tool was implemented by the students, so they were already familiar with the tool before the case study.

Next, we describe the first step of the case study by going through an example. The software component we want to measure is the distributable component, like the Truck Bed Cargo Lamp component shown in Fig. 6 and Fig. 7. The boundary, the users, the surrounding software, and any engineered devices in Fig. 1 are clearly defined by the component diagram. The distributable components are always defined at the same level of granularity and level of decomposition, which is important to be able to compare CFP values to each-other. To illustrate the usage of the UML Profile we explain how the distributable component in Fig. 6 and Fig. 7 is modeled. The result is shown in Fig. 10.

The maximum latency requirement in Fig. 7 is modeled as a required interface, because it will be implemented as a periodic invocation of the component with a maximum allowed period time. The vehicle power mode requirement in Fig. 7 is in fact a required interface, and it is modeled accordingly. The other interfaces are modeled as in Fig. 6. The CARGO LAMP PRESENT requirement in Fig. 7 is modeled as a CompSizeProperty with Direction=in. This is the needed information for COSMIC, so we can directly use the mapping rules defined in Table 1 to obtain the CFP value. The result is CFP=7, i.e. 5 Entry data movements+1 Exit data movement+1 Read data movement.

In Fig. 10 it is also shown that the CompSizeComponent stereotype has attributes to store categorization factor values, like functionality=Comf & Conv, and decomposition_level=Distributable. This information is important in order to identify groups of similar components.

The information in Fig. 10 was modeled in Rhapsody, and an XML file containing the information was exported from Rhapsody. In [30] we show a concrete example how the information in Fig. 10 was modeled in Rhapsody. The CompSize tool imported the information, identified the data movements, calculated the CFP value to 7, and identified the categorization values. This is shown in Fig. 4. The categorization factor values in Fig. 10 are used to select the proper linear regression model to convert the CFP value into bytes, and hence estimate the implemented Code Size of the distributable component. For the Truck Bed Cargo Lamp component the resulting estimated Code Size is 1417 bytes, as shown in Fig. 5. Hence, we conclude that the UML Profile can capture the information needed for COSMIC, and that the tool can import this information from an XML file.

The second step of the case study modeled 10 components based on requirement specifications, as described in Fig. 9. The purpose is to collect timing data. The time was measured for each of the following activities; reading the textual specification to understand the requirements and to identify the distributable component and its interfaces, modeling the distributable component and its interfaces, reading the textual specification a second time to identify the additional information needed for COSMIC, modeling the additional information in the UML Profile, feeding the information into the CompSize tool and obtaining the estimated bytes value.

In addition, the measured CFP values were compared to our previous measurements of the same components. The purpose is to assess whether the students have identified the majority of the data movements, and to make sure that the timing data is relevant. A scatter plot comparing the CFP values obtained in the case study and our previous CFP values from [25], is shown in Fig. 11. As can be seen, the resulting CFP values from the student measurements deviate from ours, but in general the students seem to have identified the majority of the data movements. The R^2 value from the student measurement is high (R^2=0.80), which confirms the strong correlation between CFP and bytes we have found in our measurements. We expected some deviation between the measurements performed by the students and our own measurements, because published experiments show good repeatability in CFP measured by experienced engineers, but poor repeatability in CFP measured by inexperienced

Table 2 Statistics for the relative effort for the UML profile.

Mean value (%)	Std deviation (%)	Min value (%)	Max value (%)
13	4.3	5.3	20

engineers [7]. Therefore, we concluded that the timing data is obtained from realistic estimations.

5.3 Interpretation of Results

The time for each step in the estimation process was measured and collected during the case study. The time needed for reading the textual specification to identify the additional information for COSMIC and modeling that in the UML Profile was converted into percentage of the total time.

The time for one of the components was identified as an outlier, because it was significantly larger than for the other components and the student was in fact sick at the time. The statistics of the resulting data for the remaining 9 components is presented in Table 2. Here we see the amount of added effort for the UML Profile compared to the total effort.

From interviews with architects at Saab we have obtained the manual effort involved in creating the standard Component Diagrams according to Fig. 6, to around 6 man hours for a typical distributable component. This effort includes reading of textual specification, modeling the component diagram, having review meetings with the domain expert, etc.

So if we add an additional 13% (mean value from Table 2) of effort to the 6 man hours, we would burden the architect with an additional 47 minutes (0.13×6 hours) for modeling the UML Profile. The additional 47 minutes are much less than the 2-4 hours needed for manual measurement and estimation. For the complete application software of a typical Saab car it would require around 900 man hours (0.5 man years) with our model-based and automated approach, instead of up to 4800 man hours (2.5 man years). In addition, we expect that the effort needed in our approach will decrease even further in a practical case. The reason is that an architect knows from the beginning what information to look for, and will normally read and model everything for a component at the same time. In the case study, the students were instructed to identify and model the distributable component with its interfaces first, and then the rest of the Profile in a second run.

A natural question at this point is why the architect would be willing to model the additional information for the purpose of estimating Code Size. The motivating factor is that the architect needs the estimated values as support for allocation decisions, architecture studies in early development phases, etc.

A significant difference in our model-based and automated approach compared to the manual approach, is that much of the COSMIC measurement knowledge is needed when the architect models the UML Profile for the component. This is the step where the actual measurement takes place. Therefore it is crucial that the architect receives COSMIC knowledge support, either directly from a measurement engineer or from written guidelines. Our recommendation is the latter, and that is the way we plan to implement the approach at Saab. The guidelines are important to obtain high repeatability and consistency in the process.

6. Evaluation of Validity Threats

Our case study was conducted by two Master students with no prior experience about the COSMIC method and limited knowledge about the automotive domain. This fact is likely to affect the accuracy of the estimates, as well as the absolute effort to obtain the estimate. However, in this paper we focus on the relative effort required in each phase of the estimation process. Since the students are equally inexperienced in each phase, we expect effort data that are less biased than if an architect who is familiar with parts of the process would conduct the case study. Moreover, the case study was defined, planned, supervised, and analyzed by the authors of this paper. The first author has 15 years of experience from software development activities, of which 6 years were spent managing Architecture teams at Saab and GM. This experience is important in order to obtain and analyze the results of the study.

The UML Profile is tailored to capture the information needed for the COSMIC method. We regard this as a minor limitation, because COSMIC is an approved ISO standard for measuring the Functional Size of software and much of the current publications concerning software size measurement apply COSMIC.

We have only used requirement specifications, Component Diagrams and software implementations from two automotive companies. This means we can only make conclusions that are valid in this particular domain. Therefore we plan to evaluate our approach with data from other domains.

7. Related Work

Marin et al. [31] presents a survey of existing literature related to measurement procedures based on COSMIC FP. Eleven procedures are presented of which two applies to the real-time systems domain. Of these two, the most relevant one [6] uses models developed in the ROOM (Real-time Object Oriented Modeling) language as input to the McROSE tool [7]. Their work is similar to ours, but they use another modeling language as input for COSMIC measurement. They conduct a case study to validate the tool, but they do not report on the efficiency obtained using the tool compared to manual measurement.

Soubra et al. [36] designed an FSM procedure based on COSMIC FP to measure the functional size of requirements modeled in the Simulink tool.

Other works use UML diagrams like use case, class, component, and sequence diagrams as input for COSMIC

measurement [20]–[22]. The purpose is to improve the practice of COSMIC measurement and to automate the measurement process using a tool, but the tool remains to be developed.

Another group of publications report on how to use UML models as input for IFPUG FPA measurement [8], [38]. In [38], it is shown that UML class diagrams and sequence diagrams can be used as input for a software tool that automatically calculates the IFPUG FP.

Stern [34] reports on lessons learned from using COSMIC FP for effort estimation purposes at Renault automotive company. Very strong correlation (R^2=0.93) were found between CFP and supplier effort invoice data. Stern and Gencel [35] investigate the relationship between COSMIC FP and memory size of functions using data from the automotive industry. They found very strong correlation (R^2=0.99) in a range of Functional Sizes from CFP=7 to CFP=748. This confirms our own results reported in [23]–[28].

8. Conclusion and Future Work

The goal of this paper was to investigate how the manual effort involved in estimation of Code Size can be minimized. We defined a UML Profile capturing all information needed for estimation of Code Size, and developed a tool for automated estimation of Code Size based on CFP.

We conducted a case study using requirement specifications and software implementations from the automotive industry to answer the research questions. The case study showed that the UML Profile can capture all the information needed by the COSMIC method. The case study also showed that the effort for estimating the implemented Code Size of a component is reduced from 2-4 hours to well below 1 hour. So this work illustrates how the use of UML models can save manual effort (and hence money) in a realistic case.

As future work we plan to develop written guidelines to the architects about how they are supposed to model the UML Profile to obtain accurate estimation results. The guidelines and our model-based estimation approach will be further evaluated in a case study conducted by the actual architects.

Acknowledgements

We would like to express our thanks to Tigran Harutyunyan and Tony Heimdahl for implementing and validating the CompSize tool, as well as for conducting parts of the case study in this work.

References

[1] A. Abran, J.M. Desharnais, A. Lesterhuis, S. Oligny, D. St-Pierre, and C. Symons, The COSMIC Functional Size Measurement Method, Version 3.0.1, Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761:2003), Published by the COSMIC Group, 2009.

[2] A. Albrecht, "Measuring application development productivity," Proc. IBM Applications Development Symposium, pp.83–92, Monterey, CA, Oct. 1979.

[3] A. Albrecht and J. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," IEEE Trans. Softw. Eng., vol.SE-9, no.6, pp.639–648, 1983.

[4] A. Albrecht, AD/M Productivity Measurement and Estimate Validation, IBM Corporate Information Systems, IBM Corp., Purchase, NY, 1984.

[5] R. Baillargeon and R. Flores, "From algorithms to software – A practical approach to model-driven design," SAE paper 2007-01-1622.

[6] H. Diab, M. Frappier, and R. St-Denis, "Formalizing COSMIC-FFP using ROOM," ACS/IEEE Int'l Conf. on Computer Systems and Applications, 2001.

[7] H. Diab, F. Koukane, M. Frappier, and R. St-Denis, "McROSE: Automated measurement of COSMIC-FFP for rational rose real time," Information and Software Technology, vol.47, no.3, pp.151–166, 2005.

[8] W. Fornaciari, P. Micheli, F. Salice, and L. Zampella, "A first step towards Hw/Sw partitioning of UML specifications," Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE'03), pp.668–673, 2003.

[9] C. Gencel and O. Demirors, "Functional size measurement revisited," ACM Trans. Softw. Eng. Methodol., vol.17, no.3, Article 15, June 2008.

[10] C. Gencel, R. Heldal, and K. Lind, "On the relationship between different size measures in the software life cycle," Proc. IEEE Asia-Pacific Software Engineering Conference (APSEC 2009), pp.19–26, 2009.

[11] IBM Rhapsody, http://www.ibm.com/

[12] IFPUG, Function Point Counting Practices Manual, Release 4.1, IFPUG, Westerville, OH, 1999.

[13] ISO/IEC 14143-1:2007, "Information Technology – Software Measurement – Functional Size Measurement – Part 1: Definitions of concepts," 2007.

[14] ISO/IEC 14143-2:2002, "Information Technology – Software Measurement – Functional Size Measurement – Part 2: Conformity Evaluation of Software Size Measurement Methods to ISO/IEC 14143-1," 2002.

[15] ISO/IEC TR 14143-3:2003, "Information Technology – Software Measurement – Functional Size Measurement – Part 3: Verification of Functional Size Measurement Methods," 2003.

[16] ISO/IEC TR 14143-4:2002, "Information Technology – Software Measurement – Functional Size Measurement – Part 4: Reference Model," 2002.

[17] ISO/IEC TR 14143-5:2004, "Information Technology – Software Measurement – Functional Size Measurement – Part 5: Determination of Functional Domains for Use with Functional Size Measurement," 2004.

[18] ISO/IEC 14143-6:2006, "Guide for the Use of ISO/IEC 14143 and Related International Standards," 2006.

[19] ISO/IEC 19761:2003, "Software engineering – COSMIC-FFP – A functional size measurement method," 2003.

[20] L. Lavazza and V. Del Bianco, "A case study in COSMIC functional size measurement: The rice cooker revisited," Int'l Workshop on Software Measurement (IWSM), Nov. 2009.

[21] L. Lavazza and G. Robiolo, "Introducing the evaluation of complexity in functional size measurement: A UML-based approach," Int'l Symposium on Empirical Software Engineering and Measurement (ESEM 2010), 2010.

[22] G. Levesque, V. Bevo, and D. Tran Cao, "Estimating software size with UML models," Canadian Conference on Computer Science & Software Engineering (C3S2E '08), pp.81–87, 2008.

[23] K. Lind and R. Heldal, "Estimation of real-time system software size using function points," Proc. Nordic Workshop on Model Driven Engineering (NW-MoDE), pp.15–28, 2008.

[24] K. Lind and R. Heldal, "Estimation of real-time software code size using COSMIC FSM," Proc. IEEE Intl. Symposium on Object/component/service-oriented Real-time distributed Computing

(ISORC 2009), pp.244–248, 2009.
[25] K. Lind and R. Heldal, "Estimation of real-time software component size," Nordic Journal of Computing (NJC), no.14, pp.282–300, 2008.
[26] K. Lind and R. Heldal, "On the relationship between functional size and software code size," Proc. Workshop on Emerging Trends in Software Metrics (WETSoM'10) held in conjunction with the Intl. Conf. of Software Engineering (ICSE'10), 2010.
[27] K. Lind and R. Heldal, "Categorization of real-time software components for code size estimation," Int'l Symposium on Empirical Software Engineering and Measurement (ESEM 2010), 2010.
[28] K. Lind and R. Heldal, "A practical approach to size estimation of embedded software components," IEEE Trans. Softw. Eng., no.99, 2011.
[29] K. Lind and R. Heldal, "A model-based and automated approach to size estimation of embedded software components," Proc. ACM/IEEE Intl. Conf. on Model Driven Engineering Languages and Systems (MoDELS), 2011.
[30] K. Lind, R. Heldal, T. Harutyunyan, and T. Heimdahl, "CompSize: Automated size estimation of embedded software components," Int'l Workshop on Software Measurement (IWSM), Nov. 2011.
[31] B. Marin, G. Giachetti, and O. Pastor, "Measurement of functional size in conceptual models: A survey of measurement procedures based on COSMIC," Int'l Workshop on Software Measurement (IWSM), Nov. 2008.
[32] OMG, Unified Modeling Language (UML), Superstructure Specification, V2.3, Object Management Group, http://www.uml.org/
[33] OMG, Unified Modeling Language (UML), Infrastructure Specification, V2.3, Object Management Group, http://www.uml.org/
[34] S. Stern, "Practical experimentations with the COSMIC method in automotive embedded software field," Int'l Workshop on Software Measurement (IWSM), Nov. 2009.
[35] S. Stern and C. Gencel, "Embedded software memory size estimation using COSMIC: A case study," Int'l Workshop on Software Measurement (IWSM), Nov. 2010.
[36] H. Soubra, A. Abran, S. Stern, and A. Ramdan-Cherif, "Design of a functional size procedure for real-time embedded software requirements expressed using the simulink model," Int'l Workshop on Software Measurement (IWSM), Nov. 2011.
[37] C. Szyperski, Component Software: Beyond Object-Oriented Programming, 2nd ed. Addison-Wesley Professional, Boston, 2002, ISBN 0-201-74572-0.
[38] T. Uemura, S. Kusumoto, and K. Inoue, "Function-point analysis using design specifications based on the unified modelling language," J. Software Maintenance and Evolution: Research and Practice, vol.13, no.4, pp.223–243, 2001.

Kenneth Lind received the MSc degree from Linköping University of Technology, Sweden, in 1993. He then worked one year as a system engineer at Scania Trucks and Buses AB, followed by four years as a software engineer at Volvo Aero Corp. After that he worked at Saab Automobile AB during the period 1998-2011 as a manager and project leader within the system architecture area. Since then he has been with Viktoria Institute AB and is currently a researcher within system architecture. His research interests include software estimation, software architecture, and model-driven development in the context of embedded systems.

Rogardt Heldal graduated with honors degree in Computer Science from University of Glasgow, Scotland in 1989. He obtained a Ph.D. in Computer Science from Chalmers University of Technology, Sweden in 2001 where he is a senior lecturer now. Currently he has a Vinnova grant to work with software architecture in collaboration with Saab Automobile AB. Since 2010 he also is working (80%) at Ericsson due an SSF's award for personal mobility. His main research interests include: program transformation, model transformation, security, formal and informal specification, software estimation, and model-driven development in the context of embedded systems. He has several publications in each of these areas.