PL FSM: AN APROACH AND A TOOL FOR THE APPLICATION OF FUNCTIONAL SIZE MEASUREMENT IN SOFTWARE PRODUCT LINE ENVIRONMENTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS
OF
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖNDER EREN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF INFORMATION SYSTEM

SEPTEMBER 2014

PL FSM: AN APROACH AND A TOOL FOR THE APPLICATION OF FUNCTIONAL SIZE MEASUREMENT IN SOFTWARE PRODUCT LINE ENVIRONMENTS

Submitted by **Önder EREN** in partial fulfillment of the requirements for the degree of **Master of Science in the Department of Information Systems**, **Middle East Technical University by**,

Prof. Dr. Nazife Baykal _____
Director, Informatics Institute

Prof. Dr. Yasemin Yardımcı Çetin _____
Head of Department, Information Systems

Prof. Dr. Onur Demirörs _____
Supervisor, Information Systems, METU

Instructor. Dr. Barış Özkan _____
Co-Advisor, Information systems, ATU

Examining Committee Members

Prof. Dr. Semih Bilgen _____
EEE, METU

Prof. Dr. Onur Demirörs _____
IS, METU

Assist. Prof. Dr. Aysu Betin Can _____
IS, METU

Assoc. Prof. Dr. Altan Koçyiğit _____
IS, METU

Instructor Dr. Barış Özkan _____
IS, ATU

Date: 12.09.2014

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this wok.

Name, Last name: Önder Eren

Signature: _____

ABSTRACT

PL-FSM: AN APROACH AND A TOOL FOR THE APPLICATION OF FUNCTIONAL SIZE MEASUREMENT IN SOFTWARE PRODUCT LINE ENVIRONMENTS

Eren, Önder
M.S Department of Information Systems
Supervisor: Prof. Dr. Onur Demirörs
Co-Advisor: Instructor Dr. Barış Özkan

September 2014, 98 Pages

In order to develop cost-efficient software it is crucial to measure the accurate software size. However; measuring the software size has up to now been almost entirely a manual process and, as such, is both time-consuming and prone to human error which can end up with time and money loss. Automation of this process is a must for the software developing companies to improve the quality of project and budget planning. This thesis introduces a mapping between COSMIC concept elements and UML conceptual elements and an automation tool in order to capture the information needed for functional software size measurement from UML diagrams in a component based software product line environment. The mapping and the tool combined is called PL FSM. The results obtained by manual measurement and automated measurement are compared and the results are observed to be close. As a result of this study, PL FSM

approach is validated in CBPL environment. The case studies have been carried out in embedded systems domain however the results can be generalized in other domains with other case studies in the future.

Keywords: *Functional Size Measurement, Automatic Functional Size Measurement, UML Profile, Product Line, Component Based Product Line*

Öz

PL FSM: YAZILIM ÜRÜN HATLARI İÇİN İŞLEVSEL BÜYÜKLÜK ÖLÇME YAKLAŞIMI VE ARACI

Eren, Önder

Yüksek Lisans, Bilişim Sistemleri Bölümü

Tez Danışmanı: Prof. Dr. Onur Demirörs

Yardımcı Danışman: Öğr. Gör. Dr. Barış Özkan

Eylül 2014, 98 Sayfa

Yazılım büyüklük ölçümünü doğru yapmak, düşük maliyetli yazılımlar geliştirmek için çok önemlidir. Ancak; yazılım büyüklük ölçümü çok yakın zamana kadar, zaman ve para kaybıyla sonuçlanabilecek büyük çoğunlukla manuel, zaman harcayan ve hataya açık bir süreç olmuştur. Bu sürecin otomatize edilmesi yazılım geliştiren şirketlerde projenin kalitesini ve bütçe planlamasını iyileştirmek için bir zorunluluktur. Bu çalışmada, COSMIC elemanları ile UML kavram elemanları arasında bir eşleştirme yapılmış ve bu eşleştirmeyi doğrulamak için de bileşen tabanlı yazılım ürün hatlarında gerekli olan bilgiyi UML diyagramlarından alan bir araç geliştirilmiştir. Eşleştirme ve araç birlikte PL FSM olarak adlandırılmıştır. Manuel ölçüm ile elde edilen sonuçlarla otomatik ölçümde elde edilen sonuçlar karşılaştırılmış ve sonuçların birbirine yakın çıktığı gözlenmiştir. Bu çalışmanın sonucu olarak, PL FSM yaklaşımı bileşen tabanlı yazılım ürün hattında geçerlilik kazanmıştır. Bu çalışmadaki durum çalışmaları gömülü sistemler alanında yapılmış olmasına rağmen gelecekte yapılacak çalışmalar ile diğer alanlara da genelleştirilebilir.

Anahtar Kelimeler: *Fonksiyonel Büyüklük Ölçümü, Otomatik İşlevsel Büyüklük Ölçümü, UML Profili, Ürün Hattı, Bileşen Tabanlı Ürün Hattı*

This thesis is dedicated to my beautiful love Nihan OCAK, my mom, dad and brother…

ACKNOWLEDGEMENT

I would like to thank the people in my life that have guided me when I needed direction, gave me encouragement when I was having doubts, motivated me when I started feeling overwhelmed, supported me when I needed support, and were patient with me when I had so many questions.

Let me begin by expressing my sincere appreciation to my advisor, *Prof. Dr. Onur Demirörs*, and my co-advisor, *Instructor Dr. Barış Özkan*, for their continuous guidance, support, patience and encouragements throughout my study.

I am grateful for my father *İbrahim Eren*, my mother *Ülker Eren* and my brother *Soner Eren* for their endless patience, encouragement and support throughout my educational pursuit. I am really happy and fortunate to be your son.

I would also want to thank *Nihan Ocak* for being there with me on every important moment of my life. Her support in this study means a lot for me.

Also I am thankful to the staff of Informatics Institute for their helps in every stage of the bureaucratic tasks.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CFP	COSMIC Function Point
COSMIC	Common Software Measurement International Consortium
FP	Function Point
FPA	Function Point Analysis
FSM	Functional Size Measurement
FUR	Functional User Requirement
IbD	Interface based Design
ICD	Interface Control Document
IDD	Interface Design Document
IFPUG	International Function Point Users Group
IS	Information System
ISO	International Organization for Standardization
IT	Information Technology
LOC	Lines Of Code
MDA	Model Driven Architecture
METU	Middle East Technical University
MIS	Management Information Systems
OOAD	Object Oriented Analysis and Design
PLA	Product Line Architecture
PL FSM	Product Line Functional Size Measurement
SPL	Software Product Line
SRS	Software Requirement Specification
SSM	Software Size Measurement
UML	Unified Modeling Language
XML	Extensible Markup Language

CHAPTER I

INTRODUCTION

Measurement is the starting point of science and it is the basic part in an engineering discipline; it gives an insight into the completion of an objective. If you can measure the thing you are talking about and express it with numbers than you know something about it. Software effort estimation is the process of predicting the size of a software product and it is used in the measurement of the sources you have to dedicate for a project to be accomplished. Managers have to know the accurate size of a software project to plan and manage the software development process. It is sure that the project will not be completed in the planned time and budget with the uncertain software size measurement (Pressman, 2005).

A successful project planning is not possible without an accurate software size measurement (Gencel & Demirörs, 2008). In addition, software size measurement is an extremely important process in order to perform effort and cost estimation, project monitoring, project control and quality control successfully.

Numerous software size measurement approaches have been developed to overcome measurement based management problems. Among these approaches, a family of methods which measure functionality attribute of the software have been developed (COSMIC Measurement Manual, 2014; IFPUG Function Point Counting Practices Manual, 2003). Functional size is a measure obtained by measuring software in terms of the functionality it delivers. It is among the most preferred measures since functional size can be measured from Functional User Requirements which are available at the early phases of development independent from implementation choices and decisions (Hericko, Rozman & Zivkovic, 2006). Functional size is measured from Functional User Requirements which are typically available in software documents such as Software Requirements Specification (SRS).

Functional Size Measurement (FSM) methods define a model of software that consists of generic concepts and constructs that describe software functionality. A Functional Size measurer is expected to construct the model of the software from Functional Users Requirements and then apply a set of rules given by the method to this model and finally quantify software functionality. One challenge in FSM is the elicitation, gathering, interpretation of Functional User Requirements from various resources that can have various representations and details depending on the characteristics that

pertain to the development method, specification techniques, project constraints, application domain and organizational choices in a specific development environment. When this challenge is not handled effectively, the results may lead to inconsistent, inaccurate measurements as well as the decreased value from exploitation of the results such as inaccurate estimations and decreased benchmarking opportunities (Fetcke, Abran & Dumke, 2001). Recognizing the need, various extensions, additional rules to the FSM methods have been proposed by method governing bodies and researchers in order to overcome this challenge due to abstractness of model elements which are hardly directly available in development environments (Ozkan & Demirors, 2009).

Following this, in this thesis study, an approach coupled with an FSM support tool (PL-FSM) has been proposed for the measurement of functional size from functionality specifications given in Unified Modeling Language (UML) in a product line (PL) environment which is structured in accordance with the interface based design method. UML diagrams are used to extract the COSMIC conceptual elements. COSMIC FSM method has been selected due to its growing popularity, international recognition and its soundness from the measurement theory viewpoint (Abran, 2010).

1.1. PROBLEM STATEMENT

Software reuse is crucially important for developing cost efficient software. Employing a Software Product Line (SPL) is an efficient way of increasing software reuse (Bosch, 2002). There are several SPL architectures and the component based product line (CBPL) is a good fit for the Model Driven Architecture (MDA) in software development. CBPL also supports abstraction of components. (Matinlassi, 2004).

Interface-based Design (IbD) is a software architecture methodology which is based on interfaces between callers and suppliers. IbD method allows CBPL components to be replaced with other component providing the same interfaces. Moreover, interface based design supports reusability and reliability in CBPL (De Alfaro & Henzinger, 2005). The method has been developed in the context of object oriented design and been used in a great harmony with the CBPL and UML design concepts (Cheesman & Daniels, 2000).

Components used in CBPL gets together to create sophisticated and distributed software systems and IbD method ease the integration of those components to the system (Enselme, Florin & Aubry, 2003). The user requirements of the components are defined in the interfaces of that component in CBPL structured in accordance with the interface based design method (Bate, Hawkins & McDermid, 2003). When CBPL and IbD methods are used together, development effort is significantly decreased (Sikora, Tenbergen & Pohl, 2011).

Despite the fact that there are numerous benefits of using CBPL architecture with the interface-based design approach; there are some significant difficulties employing FSM in this software development architecture. One of the difficulties is; SRS documents are not properly documented in CBPL environments because the user

requirements of the components are defined in the interfaces of a component and Interface Control Document (ICD) documents are sufficient for software developers to develop a component (Bate, Hawkins & McDermid, 2003). Another problem in this domain is that few research studies address FSM challenges in CBPL environment. Although there are many studies that address measurement from UML diagrams, these studies do not emphasize CBPL and IbD characteristics together and the predefined interfaces in components are not taken into account. The researches in the related field are given in detail in Chapter 2.5.

1.2. APPROACH TO THE PROBLEM

The aim of this study is to develop a COSMIC measurement approach and a supporting tool that derives Functional User Requirements (FURs) and functional size from UML diagrams that are frequently used in component based software product line environments which are structured in accordance with the IbD method. The scope of the study is illustrated in Figure 1.

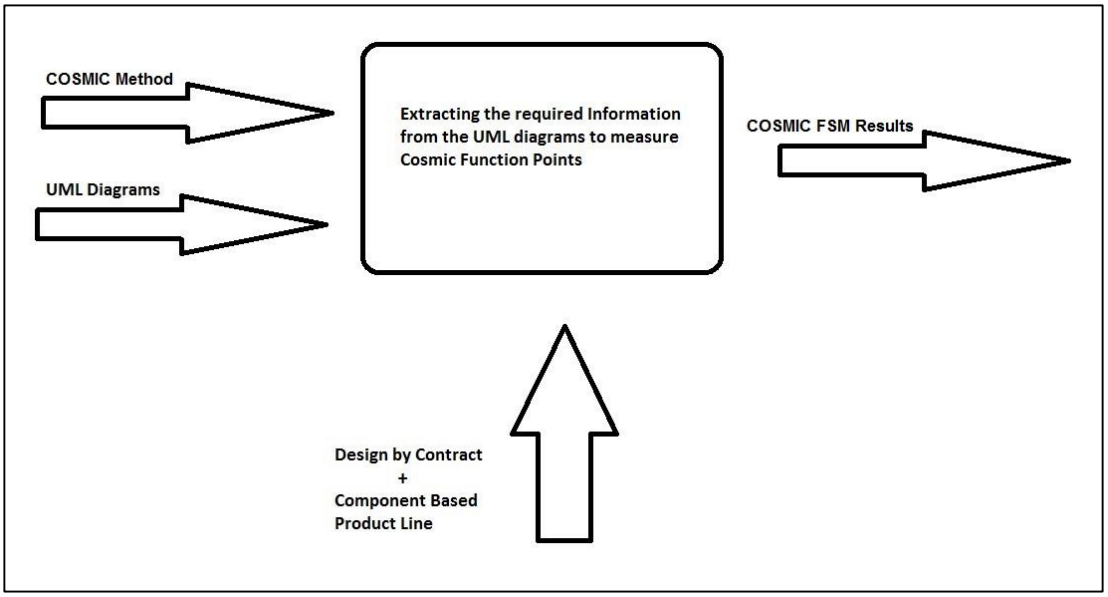

Figure 1- Scope of the Study

The approach essentially relies on the mapping of COSMIC software model concepts to the UML elements that are selected and found appropriate and intuitive for CBPL/IbD environments. Concept mapping is a frequently used step in the development of FSM techniques for specific contexts (Bianco & Lavazza, 2009). In the identification of UML elements we have investigated the question: which UML diagrams can be used to best suit the needs of FSM using COSMIC in a PL environment structured in accordance with the interface based design method?

In the study, following this mapping, the automation of COSMIC FSM is explored following the motivation that UML diagrams which provide semi-formal representation of FURs and maintained in standard data formats.

The need of automating the functional size measurement process in UML environment of a component in CBPL is obvious to decrease the error rate by reducing the human effect. The automation of the process is also valuable to decrease the time needed to measure the functional software size (Azzouz & Abran, 2004).

As a summary this study has been driven mainly by the the following research goals

- Determining the UML diagrams and diagram elements for functional size measurement in CBPL that are structured in accordance with the IbD method.
- Automating the COSMIC FSM by UML diagrams in CBPL environment.

1.3. RESEARCH ROADMAP

In this study, the need of size measurement in component based product line environments is defined first. Secondly, an explorative case study is investigated to have a better understanding of the problem statement and the problem domain. A mapping between UML elements and COSMIC concept is done to automate the functional size measurement process by developing a tool which is integrated with the IBM Rational Rhapsody UML environment. Finally, a case study is done in order to validate the mapping proposed by this study. The roadmap of the research is shown in Figure 2.

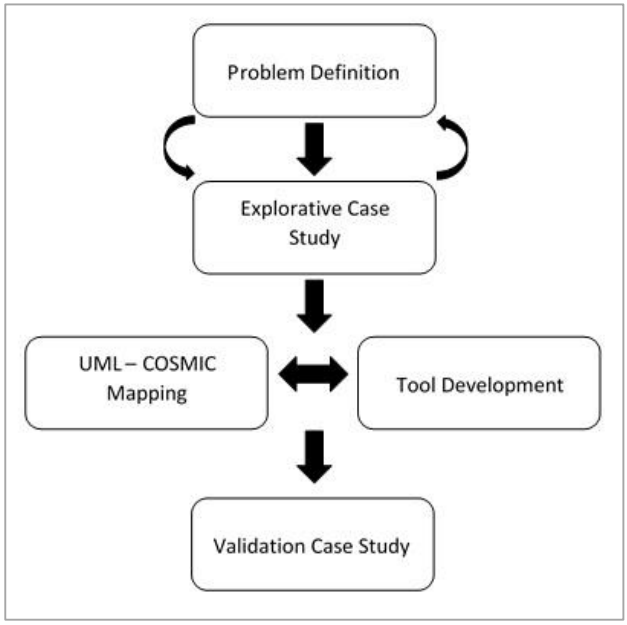

Figure 2 - Research Roadmap

1.4. OVERVIEW

Chapter 2 is the literature review in which information about the related topics and earlier studies in this field is explained. A mapping of UML conceptual elements and COSMIC concept is introduced in Chapter 3. The Product Line Functional Size Measurement (PL FSM) tool is also introduced in Chapter 3. Later on, in Chapter 4 the results of the study are investigated by a case study. Chapter 5 gives details about the contribution of the study, describes the limitations in the study, concludes the thesis with the conclusion and talks about the further research.

CHAPTER II

LITERATURE REVIEW

2.1. SIZE MEASUREMENT

Managing a project is defined as directing the tools and techniques to complete a unique, complex task taking time, budget and quality into consideration (Atkinson, Paech, Reinhold & Sander, 2001). An information about the resources needed to complete a project has a great value for project management (Farr & Nanus, 1964).

A software developing company needs to measure the software it developed or it is going to develop because it has to know its processes and products to compare its own performance with the market and to improve the effectiveness and efficiency of its operations (Dekkers, 2005). Size measurement is valuable for managers to develop cost efficient software products if it is a quick procedure and when it gives accurate results (Farr & Nanus, 1964). On the other hand, the software industry's lack of estimating the development cost, effort or time is a common known issue. The deviations between the reality and the planned estimation mainly comes from the over optimistic estimates, user changes or misunderstandings (Molokken & Jorgensen, 2003).

Hericko, Rozman and Zivkovic (2006) declared that software size measurement is a challenging task which requires a methodical approach. The types of size measurement methods are presented in the following section.

2.1.1. Types of Size Measurement Methods

Software size is a crucial measure for the objective evaluation of software engineering characteristics such as productivity and quality. Following gives a brief overview on software size measurement concepts and some common techniques.

2.1.1.1. Lines of Code

There are several types of software size measurement methods and one of them is the "Line of Code" based size measurement and it has been in use for over 50 years. It is based on counting the lines of code of accomplished projects (Hastings, 2001). But is it reliable to measure the size with Line of Code? Programmers' coding style and the language they use to develop software may change the number of Line of Code (LOC).

Vickers (2003) declared that the LOC measure could only be a comparison with the factors remaining constant such as the coders and the programming language.

Bhatt, Tarey and Patel (2012) stated that LOC is almost the first size measurement technique which basically depends on counting the lines in the source code. Line of Code is a physical entity which can easily be automated. It is in fact an indication for the size of the software but the problem is it does not really represent the productivity. LOC has many disadvantages such as:

- It depends on the language which the code is developed.
- It depends on the skill of the developer. A skilled developer may have less lines of code compared to a new developer.
- Since its input is the source code it is impossible to have a measure at the beginning of the project (in analysis or design phases).
- What to count is still a controversial issue. What are included in the source code file and what are not?
- The project which has been coded in different languages also is not suitable for LOC method.

After the evolution of the object oriented development approaches and UML usage in embedded software projects LOC technique has become inefficient and because of the drawbacks explained LOC is not accepted as a productivity measurement (Bhatt, Tarey & Patel, 2012).

2.1.1.2. Function Point Analysis

The Function Point Analysis method was developed by Alan Albrecht in 1979 to measure the size of a business information system (Symons, 1988). Rather than counting the lines of code, Function Point Analysis (FPA) focuses on system functionality. FPA is one of the most effective and widely used methods

of software size measurement (Hastings, 2001). The rules were clarified and the method was improved by the International Function Point Users Group (IFPUG) in 1984. A standardized methodology to measure the size of a software application was provided by the FPA method. From the user point of view, the functionality which is mainly user's requests and receives are measured by the FPA method. Function Points are defined by Albrecht as "a dimensionless number defined in function points, which we have found to be an effective relative measure of function value delivered to customer" (Abran & Robilland, 1996). Boehm and DeMarco (1997) has declared that function point calculated was independent of the language on which the code is developed, development methodology or the skill of the developer.

The application boundary and five types of components; three types of elementary processes which are input, output, inquiry and two types of interface files which are logical and external has to be determined in Albrecht's model of functional point analysis. When these five types of components are identified, they are then weighted for complexity and are given unadjusted function point which is called as UFP. The total of 'UFP's for all components is then multiplied by a Value Adjustment Factor (VAF) which is defined as fourteen General System Characteristics. The mechanism of Albrecht (IFPUG) 'Function Point' model is illustrated in Figure 3 (Symons, 2001).

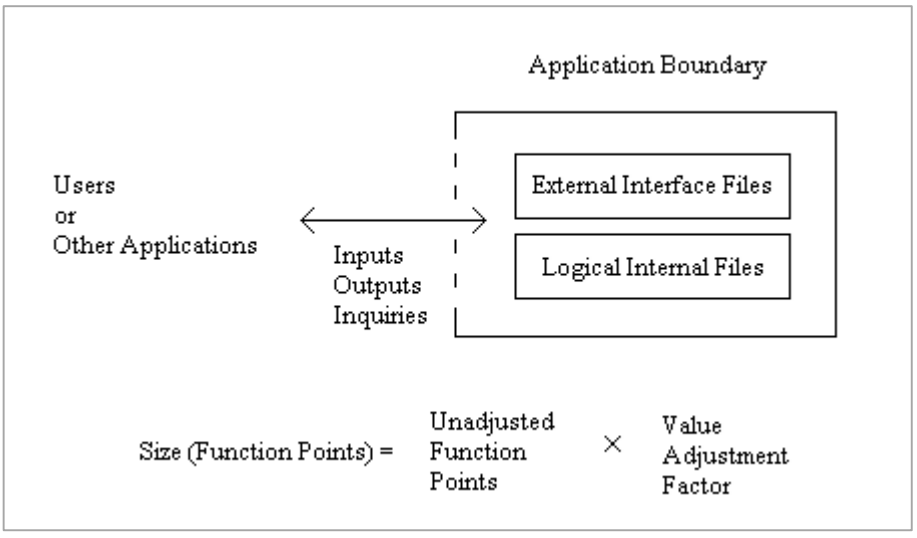

Figure 1 - The Albrecht (IFPUG) 'Function Point' model

Albrecht's FPA method was refined in 1988 (Release 2.0), 1990 (Release 3.0), 1994 (Release 4.0), 1999 (Release 4.1) and 2005 (Release 4.2) by IFPUG, however it was declared that they were consistent with Albrecht's original method. Boehm (1997) stated that it was still very close considering the two decades that have been elapsed since Albrecht's original publication.

The modifications made to the Albrecht's FPA method were basically to eliminate inconsistencies for determining function points (FP), and adapting to new technologies such as GUI elements and Object Oriented Design. IFPUG, MK II which were set up in 1988 and COSMIC in 1999 are the three notable groups which are still working on refining FPA method (Symons & McGarry, 2001).

FPA method has been adapted to object oriented models by taking UML standards into consideration (Lehne, 1997). FPA method had also some limitations such as:

- Measurement is subjective to the person who evaluates the method
- Gaining proficiency in FPA method is not easy
- The procedure is time consuming

COSMIC method, details of which is given in section 2.2, was selected as the FPA method to be automated in this study because of the fact that it is well defined, suitable for the embedded software and has many studies to automate this method in UML environments.

2.1.2. COSMIC Functional Size Measurement Method

The COSMIC group was founded in 1998 to propose a new type of size measurement method. COSMIC was first introduced by The Common Software Measurement International Consortium as a new version of FP method in 1999. The weaknesses and strengths of the earlier used methods such as IFPUG FPA, Mk II FPA, NESMA FPA and version 1.0 of the FSM method was defined by the COSMIC group (Oligny, Abran & Symons, 2000). COSMIC FSM has been approved as a functional size measurement method by International Organization for Standardization (ISO) for sizing software based on their functional user requirements (Poels, 2003). The COSMIC group intended to develop a new Functional Size Measurement Method which could be used in both embedded and business application software (Abran, 1999). However, the COSMIC method is not designed for measuring the complex mathematical algorithm including software. It is also declared in COSMIC FP Measurement Manual (2003) that the method was not applicable to simulation software, self-learning software and weather forecasting systems.

The COSMIC method is about applying a set of rules, processes and principles to the Functional User Requirements (FUR) of the software to be measured which outputs a numerical value representing the functional size of the software. In the COSMIC Measurement Manual (2014) it was declared that the functional size measured by the COSMIC method was independent of implementation decisions whether the software was embedded or not.

The COSMIC measurement process is consisted of three phases which are the Measurement Strategy, the Mapping Phase and the Measurement Phase. The result of applying these processes to the software to be measured is CFP called COSMIC Function Point. Figure 4 illustrates the COSMIC measurement process, the inputs and the outputs of each phase (COSMIC Measurement Manual, 2014).

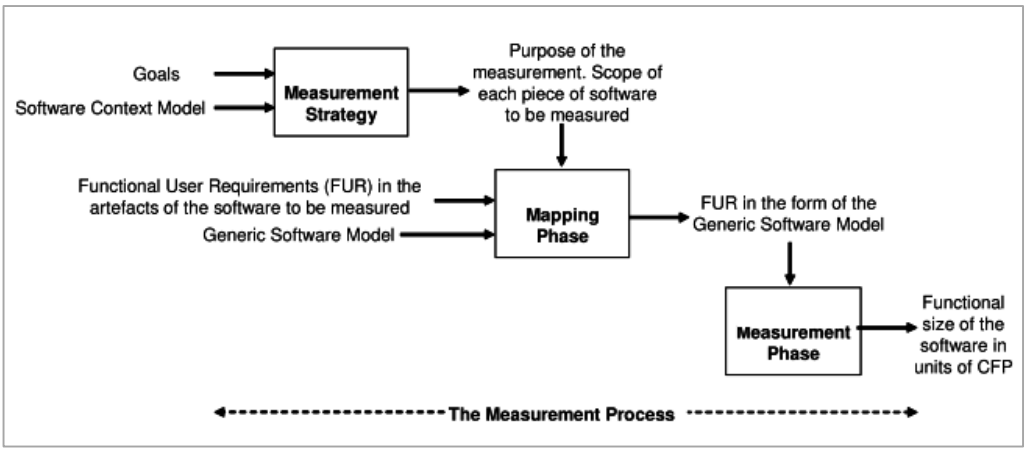

Figure 2 - Structure of the COSMIC Method

As shown in Figure 4 before starting a COSMIC measurement it is compulsory to define the purpose and the scope of the measurement. Software boundary is also defined in this Measurement Strategy step. Data groups and functional processes are identified in the mapping phase. In the measurement phase data movements which are counted as 1 CFP are identified. Data movement types and their relationship with the functional process and data groups are shown in Figure 5.

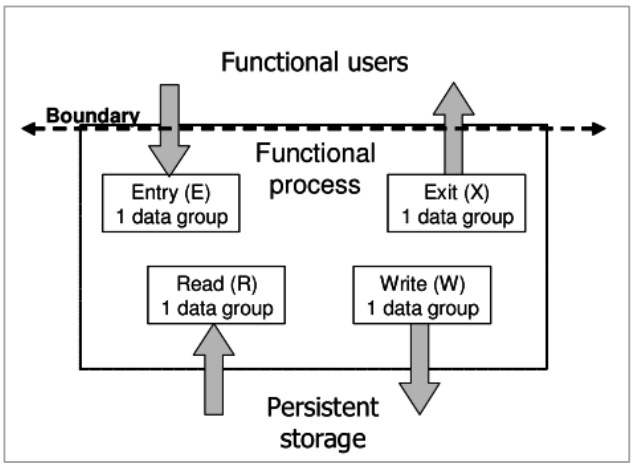

Figure 3 - Data Movement Types

- An Entry (E) moves a data group from a functional user across the boundary into the functional process where it is required.

- An Exit (X) moves the data group from the functional process across the boundary to the functional user where it is required.
- A Read (R) is a data movement that moves a data group from persistent storage to functional process where it is used.
- A Write (W) is a data movement that moves a data group from the functional process to the persistent storage where it is stored.

To calculate the CFP the numbers of the data movements are counted in each functional process. The functional sizes of each data movement type are then added up to have a single functional size.

$$\text{Size (functional process}_i) = \Sigma \text{ size (Entries}_i) + \Sigma \text{ size (Exits}_i) + \Sigma \text{ size (Reads}_i) + \Sigma \text{ size (Writes}_i)$$

2.2. SOFTWARE PRODUCT LINES

SPL is a software engineering methodology for creating a collection of software products from a repository of software assets. Developing cost efficient software products is highly related with the software reuse paradigm. Implementing a Software Product Line (SPL) is proven to be an efficient method for increasing software reuse (Bosch, 2002). Reducing software cost and keeping up with the project plan while increasing the product quality is possible by a SPL (Clements & Northrop, 2001). In a software product line, all software components are collected in a configuration management tool after the component is validated.

Most software developing companies provide products for a particular market, thus the software they develop have much in common (Voelter & Groher, 2007). These software developing companies are investing in software product line architecture to respond quickly to the requirements of the customers. Product Line based architecture gives them ability to develop new products faster and easier with more quality (Dikel, Kane, Ornburn, Loftus & Wilson, 1997). Software Product Line architecture help these companies shorten the development procedure, increase the percentage of reused components and stay competitive in the market.

The success and effectiveness of a SPL approach is highly related with the early identification of the commonalities of the products and the management of the feature variability within the portfolio. The flexibility to adapt to new product requirements ability in the SPL is created in domain engineering. In application engineering the assets created in the domain engineering process are used to develop the software products. Products differ with the requirements of the customer that defines which of the features will be included. A feature is an additional functionality provided by one or more components of the SPL (Voelter & Groher, 2007).

There are some measures to evaluate and manage a software product line such as productivity, time to market and trends in defect density. These measures are valuable for product line management. In order to obtain these measurement results,

functional size measurement of the components in that SPL is crucial (Zubrow & Chastek, 2003). Kiebusch, Franczyk and Speck (2005) states that the management of a software product line depends on the functional size of the components located in the product line.

The functional size measure of a component in SPL is also used as a morphological characteristic to reveal the quality of the product line's architectural design (Rahman, 2004). In the past few years product line architectures have been under attention in the software research community. There are five Product Line Architecture (PLA) methods widely used which are COPA, FAST, FORM, KobrA and QADA (Matinlassi, 2004).

Since the present study focuses on component based product lines the details of CBPL are given in the next section.

2.2.1. Component Based Product Lines

Component based approach in software development increases the level of reuse significantly. Component based method supports "reuse in small". Instead of reinventing the wheel, the reusable components are changed according to the customer needs. In a specific domain the components used in software development is mostly the same. Instead of starting from scratch in a component based product line the components can be replaced easily (Atkinson, Bayer & Muthig, 2000).

In component based approaches designers concentrate on defining interfaces between the software system and the component. Software developer can implement the component in any appropriate technology as long as it supports the operations of the interface. Likewise, the users of the components can use the component by referencing the interfaces between itself and the component. This type of usage improves flexibility of the software as the component changes or replaced (Brown, 2000).

Unified Modeling Language supports component based approach and CBD is also influenced by the constraints of the UML.

2.2.2. Interface based Design

Interface-based Design is a software architecture methodology which is based on interfaces between callers and suppliers designed by Meyer in 1992. The interfaces between the modules of a software system is similar to the communication rules amongst humans or companies (Jezequel & Meyer, 1997) (Brown, 2000).

It has been developed in the context of object oriented programming and it has been used with the component based product lines in a great harmony (Garion & Torre, 2003). Interface-based Design method also suits with the UML design concepts. The method is a very challenging idea for designing abstract boxes that function in

a harmony to achieve a common goal by relying on interfaces. Interfaces are well defined rules among distinct types of components. They enhance operations of an interface with conditions. The user of the component obeys the rule of the specified interfaces but the user does not know what operations are going on in the boundaries of the component (Cheesman & Daniels, 2000).

The idea behind the interface-based design method is simply to fulfill the requirements by previously tested and used product line components. The components which are coded by obeying the restrictions of the interface-based design method can be replaced with another component providing the same interfaces (Brown & Wallnau, 1998).

Two components communicating by the interface-based design method are shown in Figure 6.

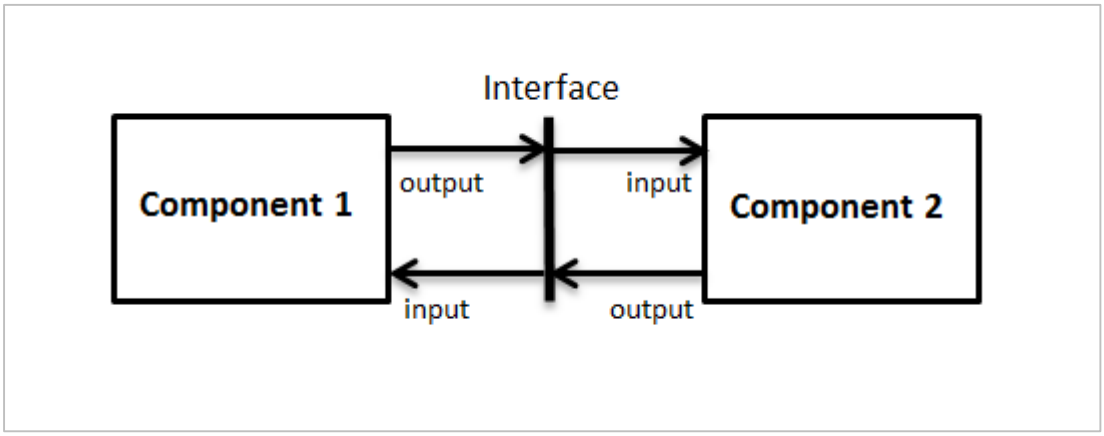

Figure 4 – Interface-based Design

Interface-based Design method provides encapsulation which hides the implementation in the component boundary. The user of the component does not need to know how an operation is implemented. The user of the component tells only what it wants from the component by obeying the interfaces (Breivold & Larsson, 2007).

The IbD method also has other benefits for the users in test domain. IbD helps software developer to decompose a system into manageable parts. The users of the component are not affected by the changes in the component. Once the component is tested and the interfaces are validated the component is ready for adding it to the product line (Meyer, 1997).

2.3. COMPONENT BASED PRODUCT LINE AND INTERFACE BASED DESIGN RELATION

Interface-based Design method is useful in designing components precisely by defining their interfaces. At the end of the design the created interfaces are independent

of each other. The implementation of the operations in these interfaces is encapsulated within the related components. The communication between the components is provided only by these interfaces. Therefore the components in each side of the communication can easily be replaced by another component providing the same interface with the replaced component. Interface-based Design supports reusability and reliability in component based software product lines (Brown & Wallnau, 1998).

Components used in the software product line gets together to create sophisticated and distributed software systems. IbD method simplifies the integration of these components to the system and the replacement of the component with another (Enselme, Florin & Aubry, 2003).

Interface-based Design method allows software developers to divide the user requirements for each component in the product line. Since the user requirements are specified in the interfaces the requirement of that component is coded in its interface (Bate, Hawkins & McDermid, 2003).

Sikora, Tenbergen & Pohl states that instead of defining system requirements, using component requirements remarkably relieve the development process and decrease the development effort. In embedded system architectures the predefined interfaces in the components has a natural link with the requirements of that component (Sikora, Tenbergen & Pohl, 2011).

2.4. UNIFIED MODELING LANGUAGE

Unified Modeling Language (UML) is a standard modeling language used for design and analysis of the software. In order to share a common understanding between the client and the developer UML contains a number of diagrams. These UML diagrams help to visualize the implementation of the software and the scenarios (Cantor, 1998). Standardization is achieved by using UML as a modeling language. UML is also available for the SPL because of its standard extensions (Clauss, 2001). The advancement and improvement of UML is controlled by the Object Management Group.

In the object oriented world modeling is extremely important. The constructed model helps the developer to get rid of the complexity of the problem details in the real world. UML is basically accepted to be the standard notation for Object-Oriented Analysis and Design (OOAD). UML is valuable for visualizing, specifying, contracting and documenting the fundamentals (requirements, architecture and design) of a software system (Booch, Rumbaugh & Jacobson, 2005).

2.4.1. UML Diagrams

In software design and analysis UML is widely accepted amongst software developers. Sophisticated and various CASE tools are designed for complex software to provide a user friendly environment for UML diagrams. UML diagrams provide the developers and clients to communicate on a problem in a visualized

environment. UML diagrams also help the developers to notice the inconsistencies and redundancies in the project (Berardi , Calvanese & Di Giacomo, 2005).

UML diagrams are mainly divided into two categories, structure diagrams and behavior diagrams. Structure diagrams illustrate the structure of the software and emphasize the elements that are crucially important in the design phase of a project, such as objects, relations between the components and instances. Behavior diagrams explain the behavior of the software visually and give details of the system scenario (OMG Unified Modeling Language, 2006).

The diagram categories and types are shown in Figure 7.

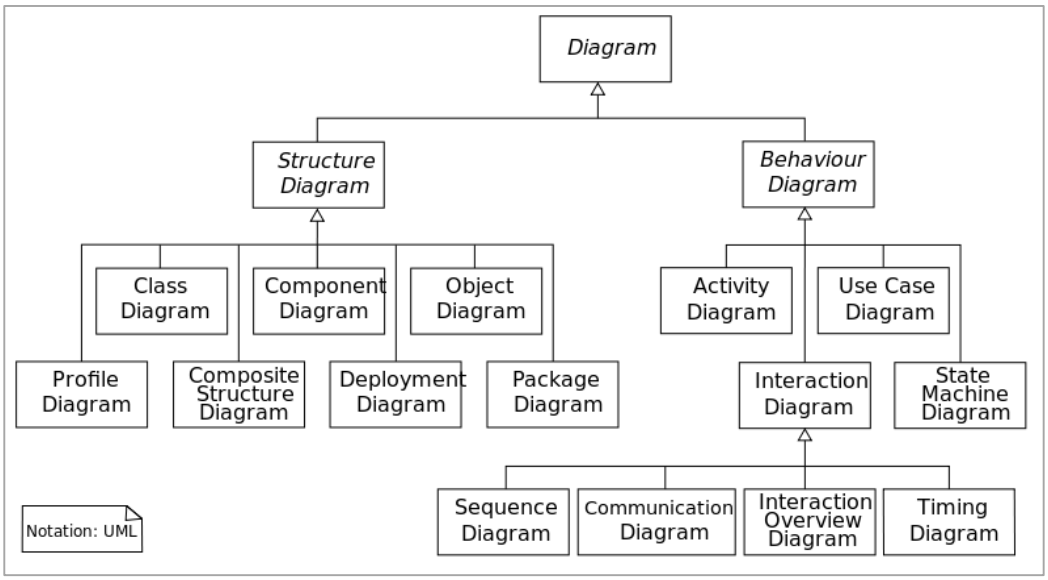

Figure 5 - UML Diagrams

2.5. DISCUSSION OF LITERATURE REVIEW

Based on the literature review, frequently used size measurement methods are time consuming and needs expertise to have a reliable result. Practically, in most cases in the market software developing organizations should respond quickly to the customer needs by estimating the size of the software (Hericko, Rozman & Zivkovic, 2006).

Product Line Software Engineering helps software developers to reuse the software and not to start from scratch (Atkinson, Bayer & Muthig, 2000). In a component based software product line the requirements of a component are specified in the component's interfaces (Sikora, Tenbergen & Pohl, 2011).

There are a number of studies investigated to measure the COSMIC function points by using the UML diagrams. Bévo, Lévesque and Abran (1999) used the UML version 1.0 to extract needed COSMIC elements from the UML diagrams of management information systems (MIS). They used the use case diagrams to find the boundary of the system to measure and sequence diagrams to find the data movements and

functional processes. They accepted each sequence diagram as a functional process. They also developed an instrument to apply the rule called Metric Xpert. They applied their method in five MIS software components and checked with the same components with the manually examined results. They found out differences fluctuating between 11 to 33 percent. In our study the domain is the embedded software components.

Nagano and Ajika (2003) used xUML in the real time systems domain to measure COSMIC function points. They used class diagrams, collaboration diagrams and state chart diagrams to identify the COSMIC elements. They verified their method with the rice cooker case study example. They compared their automated approach with the manually calculated expert results and had an error rate of 53 percent.

Azzouz and Abran (2004) used UML diagrams to measure COSMIC function points in the domain of MIS. Azzouz and Abran stated that layers cannot be determined by UML diagrams. They used stereotyping to identify the triggering events. Sequence diagrams and use case diagrams are used to identify the COSMIC elements. They developed a tool which is integrated to the Rational Rose UML tool. The verification is not done with the samples from the real world but with the rice cooker sample.

Levesque, Bevo and Cao (2008) also made a study for calculating COSMIC function point by using UML diagrams. They used sequence diagrams to count the UML messages exchanged to estimate the COSMIC function points and use case diagrams to identify the functional users. They used UML version 2.0. They also checked their method in the rice cooker example and found out an error rate 8 percent. Moreover, they stated that they should check the method with the samples from the industry.

Lavazza and Bianco (2009) used the UML diagrams to measure the well-known rice cooker example. They used the use case diagram and component diagram to find the functional users and sequence diagrams to find the entry, exit and functional processes. They identified the software boundary from the functional user requirements. In the discussion part they also exclaimed that they should use more realistic components to measure and verify their methodology. It is pointed out by themselves that consistency check is also not done in their study.

Soubra, Abran, Stern and Cherif (2011) mapped the COSMIC concepts to the Simulink conceptual elements in a real time environment. They have not developed a tool in their study but provided a basis to develop an automation tool for Software Size Measurement (SSM) in Simulink environment.

Furthermore, it is observed that there are not so many researches made about the size measurement in a product line environment. However, it is believed that software size measurement would be easier in a product line environment that is designed by the architecture interface-based design and give reliable and faster results since the inputs and outputs of the component can be extracted from the frequently used UML diagrams. In the similar studies consistency check is ignored. Most of the studies are in the domain of MIS however there are not so many researches in the field of real time systems. The studies in the literature mostly evaluate their automated methods

with the rice cooker example and not with the real software used in the industry and the partition set is small.

In this study, an automatic size measurement method based on COSMIC size measurement method in a product line environment is developed by observing the UML diagrams of a component. The study is carried out in the real time domain. In the beginning of the study the user habits in UML are interviewed with the experienced users of the software product line and interface-based design method. The size of the software product line components are measured with the requested automatic method. The results are compared with the COSMIC method's results which are calculated manually by a certified COSMIC measurement expert. The error rate is explored between these measurements. Consequently, interpretation of the data obtained from this study will bring to light if the method developed is suitable for measuring the COSMIC function points for the components in a product line environment that is structured with the interface-based design method.

Moreover, an exploratory study to examine if there is a relation between the COSMIC function point and the number of elements in a components interface is investigated. The exploratory study is an estimation approach to the COSMIC method.

CHAPTER III

PL FSM

This study has the following goals

- Determining the UML diagrams and diagram elements for functional size measurement in CBPL that are structured in accordance with the IbD method.
- Automating the COSMIC FSM by UML diagrams in CBPL environment (PL-FSM)

Following these, in the first section of the chapter, COSMIC concept and UML elements mapping specific for CBPL environments that is structured in accordance with the IbD method is given. The mapped elements are given in detail with their illustrations and explanations. In section two, the automation tool based on the mapping is introduced.

3.1. PL FSM MAPPING

In order to develop a mapping between COSMIC conceptual elements and selected UML diagram elements, a survey was conducted with experienced software developers who have UML and SPL experience and work in CBD projects. Survey results indicated that experienced developers who have experience with SPL, IbD and UML agree that sequence and composite structure diagrams are sufficient for capturing COSMIC conceptual elements. Detailed results of the survey are given in Chapter 4.2. Earlier UML and COSMIC mappings in the literature were also taken into consideration (Lind, Heldal, Harutyunyan & Heimdahl, 2011).

The COSMIC and the UML concepts are mapped in Table 1. Basically the FP obtained in the final step of a COSMIC method is calculated by counting the four types of COSMIC software model elements which are Entry (E), Exit (X), Read (R) and Write (W). The entry, exit elements are the events and functions in the required

and provided interfaces of the component due to the mapping given in Table 1. Read and Write elements are the exchanged data attributes from or to hardware or a database in a functional process.

Table 1 - COSMIC UML Mapping

COSMIC	UML Concept
Software Boundary	Boundary of the component (the component's composite structure diagrams boundary).
Functional User	Interfaces located at the ports of the component (the interface instances in the sequence diagram).
Functional Process	A set of data movements exchanged between the functional user and the software to be measured to complete a task (each sequence diagram is accepted as a functional process).
Triggering Event	Incoming message to the software boundary that starts a functional process (the first drawn arrow element of the sequence diagram).
Entry	The functions and events in the provided interface of the component that are used in the sequence diagram.
Exit	The functions and events in the required interface of the component that are used in the sequence diagram.
Read	The referred attributes in a functional process that are the arrows going out of a database instance in the sequence diagram.

Write	The updated attributes in a functional process that are the arrows going in to a database instance in the sequence diagram.

The UML diagrams that are used to extract COSMIC elements are summarized in Table 2 below. Some of the COSMIC elements are extracted by only one UML diagram where the others are captured with the collaboration of the two UML diagrams.

Table 2 - COSMIC Elements extracted in UML Diagrams

COSMIC Elements	**UML Diagrams**	
	Sequence Diagram	**Component Diagram**
Software Boundary	X	X
Functional User	X	X
Functional Process	X	
Triggering Event	X	X
Entry	X	X
Exit	X	X
Read	X	
Write	X	

The details of the COSMIC element and UML mapping are given under the subsections below.

3.1.1. Software Boundary

The software boundary is the conceptual frontier between the selected component to be measured and the rest of the software. Defining the software boundary accurately helps the measurer to determine what is to be measured and what is not. The object model diagram of the whole software is shown in Figure 8. The component marked with a red circle is selected to be measured by the measurer. The circle shows the software boundary and rest of the components are left out of the scope.

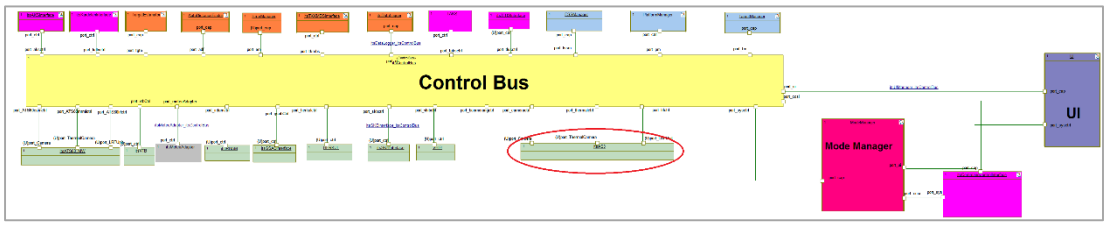

Figure 8 – Software Boundary

The composite structure diagram shown in Figure 9 below is the detailed diagram of the selected component in Figure 8. The ports are located in the software boundary and there are the pre-defined interfaces within them. The software developed to automate the measurement gets the software boundary as an input.

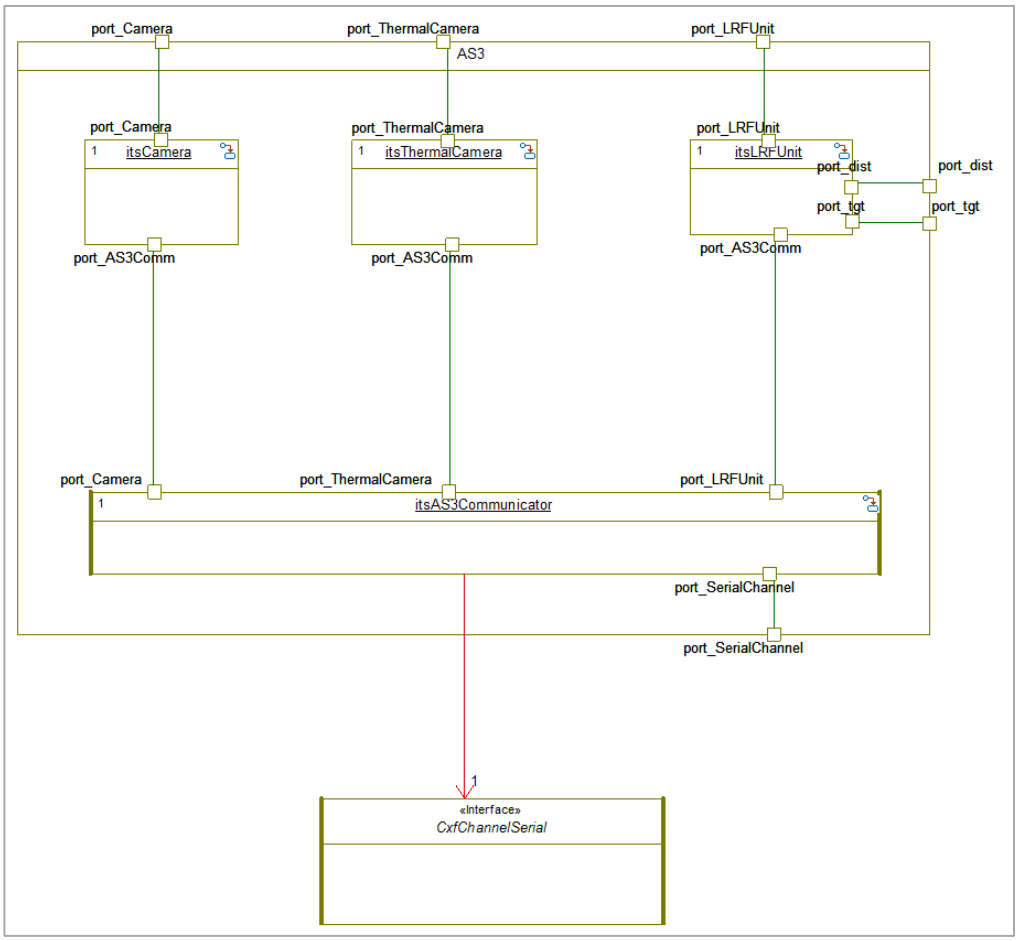

Figure 9 - Composite Structure Diagram to define Software Boundary

When the package of the component to be measured is selected by the measurer the developed software defines the software boundary automatically. Selection of the software boundary is shown in Figure 10.

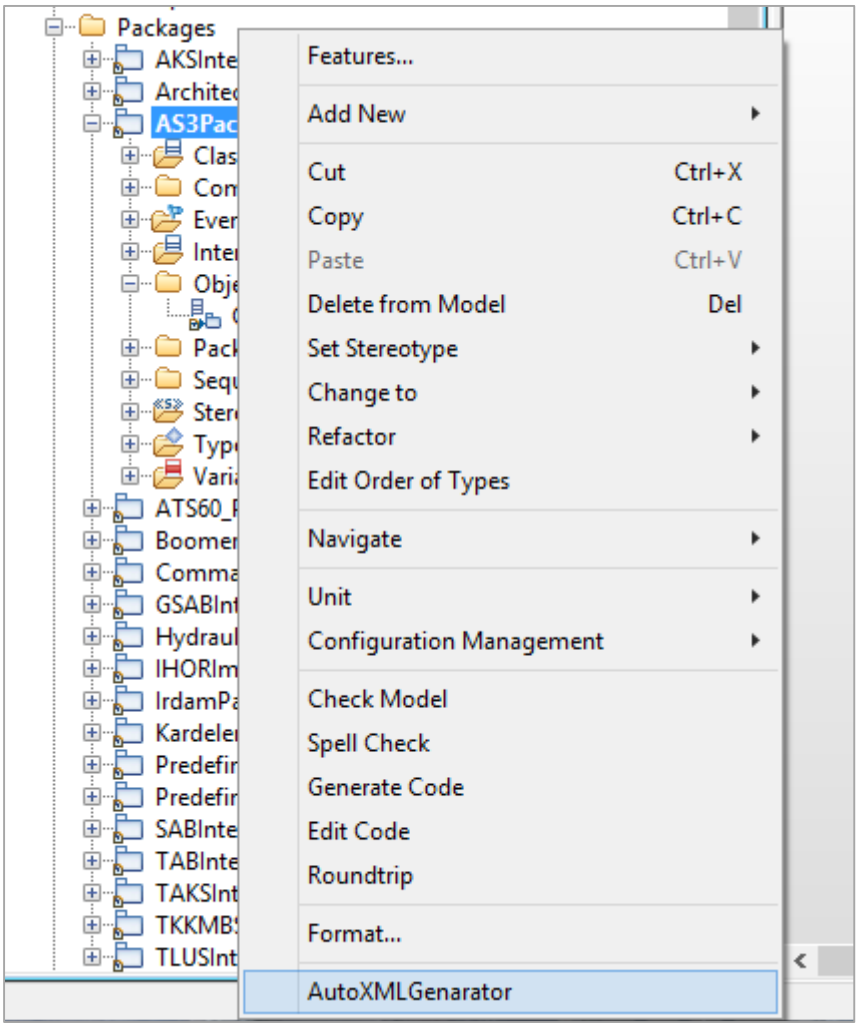

Figure 10 - Selection of the Software Boundary

3.1.2. Functional User

Functional user is the rest of the software that gives the inputs to the software to be measured and waits for the outputs produced by the component to be measured. In Figure 11 the ports that are placed in the software boundary are shown. The links are circled in red between the selected component and the external components. These external components provide the inputs to the software to be measured and require the outputs from it.

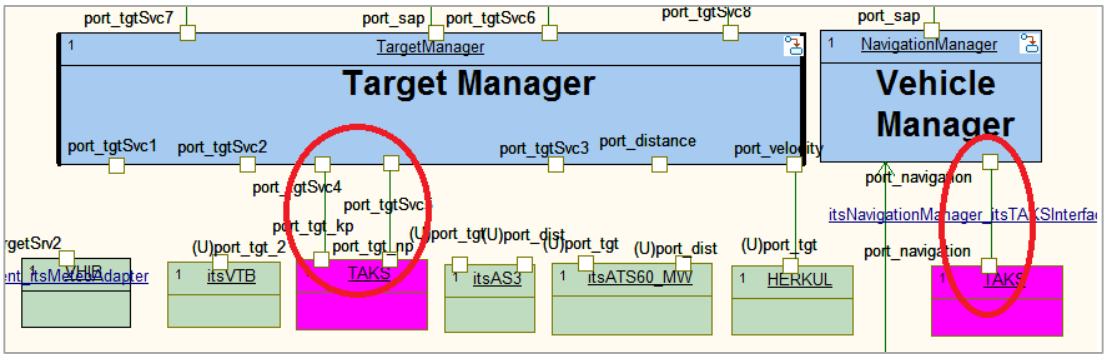

Figure 11 – Links of a Component

Since the components are the product line components they are able to be used in any project where their predefined interfaces are obeyed. The external user of the component to be measured in the product line cannot be identified but the interfaces are accepted as the external users. The functional users that are the interfaces of a component are shown in Figure 12.

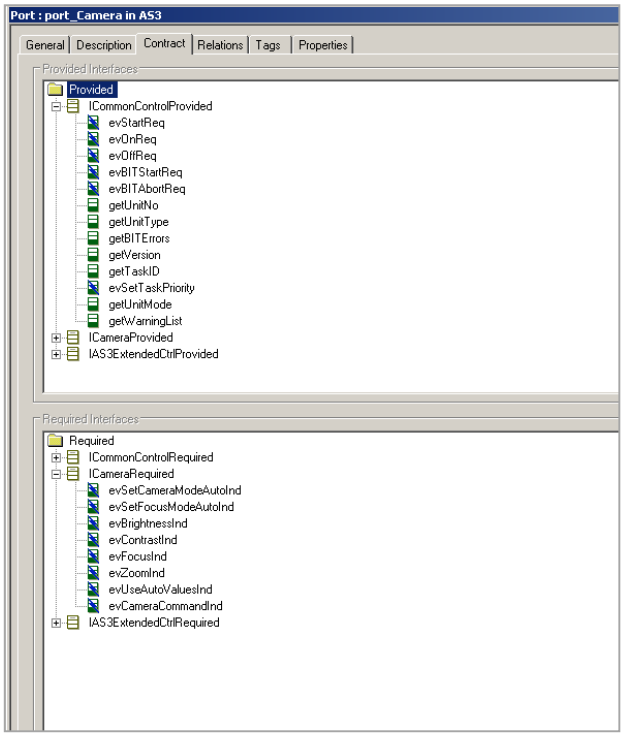

Figure 12 – Functional Users in the Composite Structure Diagram

The functional users are detected by the plug-in together with the composite structure diagram and sequence diagram. The functional users in the sequence diagram are marked in the Figure 13.

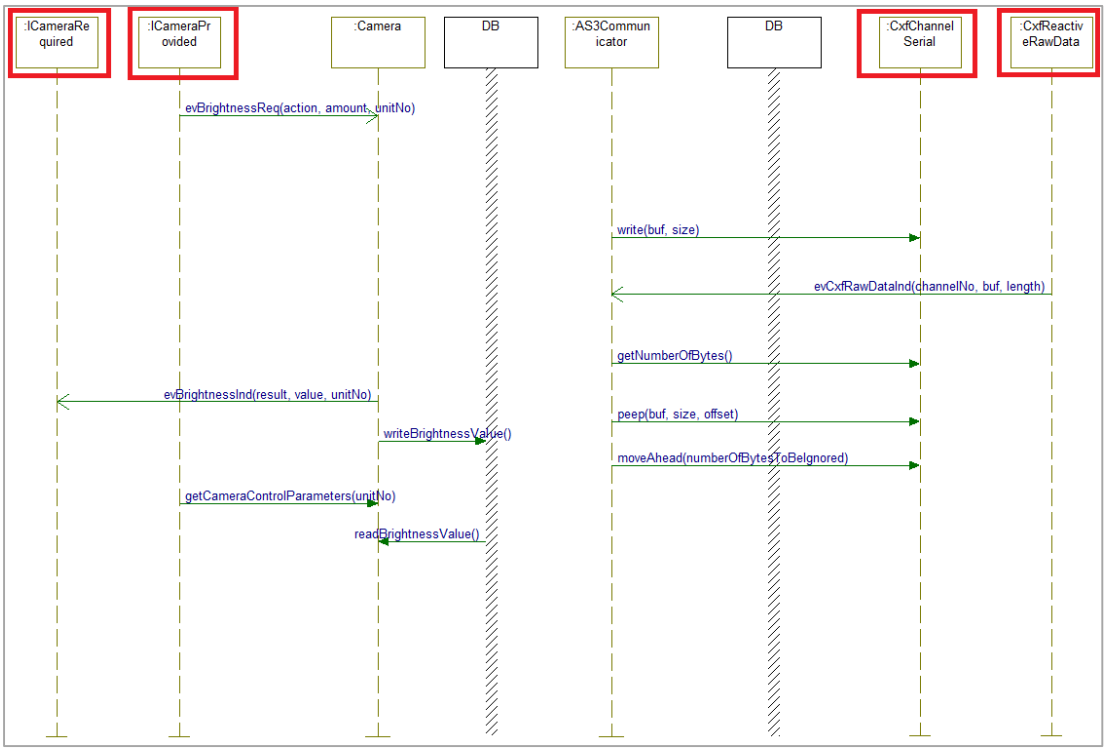

Figure 13 – Functional Users in the Sequence Diagram

3.1.3. Triggering Event

The event started a functional process is the triggering event in the sequence diagram. The designer should draw the triggering event of the functional process first otherwise the triggering event will be detected false by the automated measurement plug-in. The triggering event in a functional process is illustrated in Figure 14.

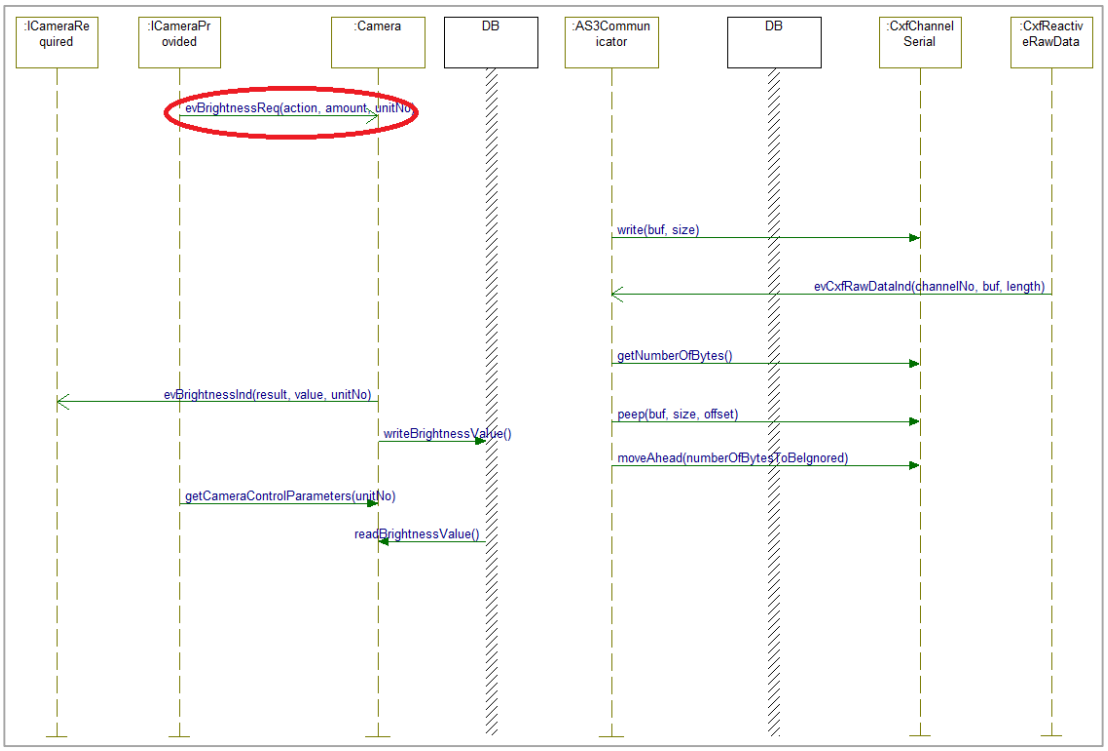

Figure 14 – Triggering Events in a Functional Process

3.1.4. Data Movements

In COSMIC, the basic functional components are data movements. COSMIC Function Point (CFP) is calculated by adding up each data movement counts. Data movements can be of four types that are Entry (E), Exit (X), Read (R) or Write (W) (COSMIC Measurement Manual, 2014). In the subsections, how the process of capturing the data movement types from the UML diagrams are explained.

3.1.4.1. Entry Data Movement

The functions and events in the provided interface of the component's ports are the Entries (E) to the software boundary. In Figure 15 the interfaces of the port is shown. The events and functions in the interfaces of the component are the data movements entering from the external components to the software boundary.

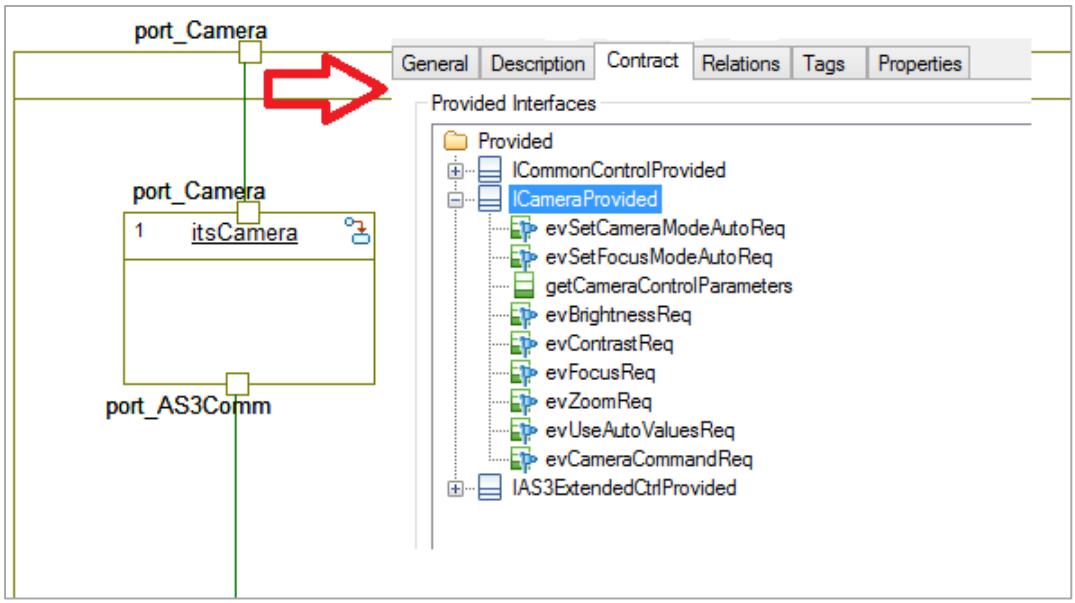

Figure 15 – Entry Data Movements

The entry data movements are captured from the sequence diagram. The entry data movement in a sequence diagram is shown in Figure 16. The entry movement has been sent by an external user of the component.

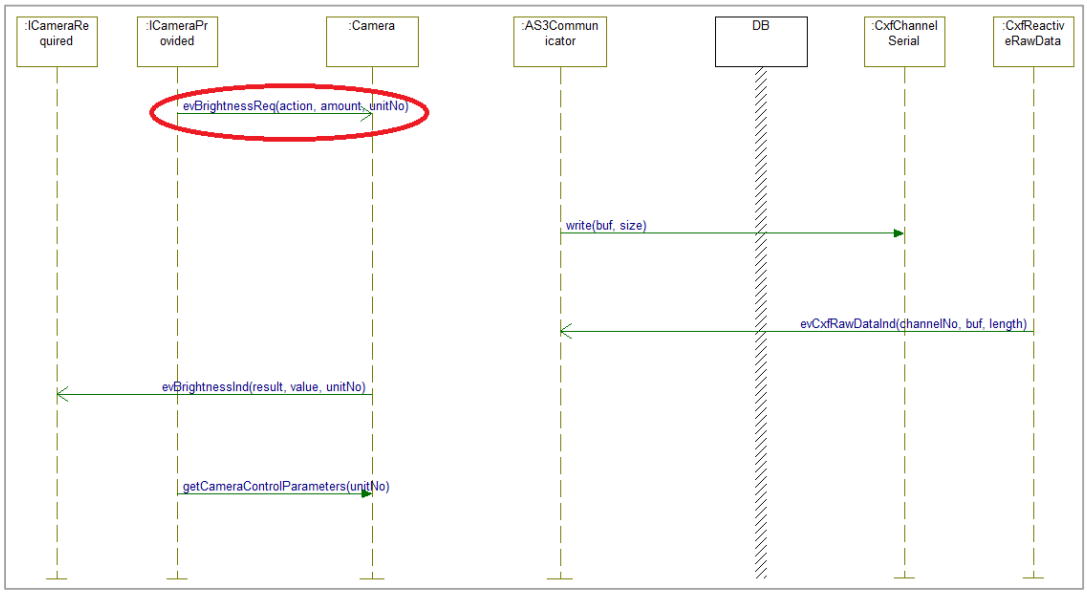

Figure 16 – Entry Data Movement in Sequence Diagram

In order to check for consistency the sequence diagram element that is counted as an entry is controlled from the interfaces of the component located in the non-behavioral ports of the composite structure diagram.

3.1.4.2. Exit Data Movement

The functions and events in the required interface of the component's ports are the Exits (X) from the software boundary. In Figure 17 the interfaces of the port is shown. The events and functions in the interfaces of the component are the data movements exiting from the software boundary to the external components.

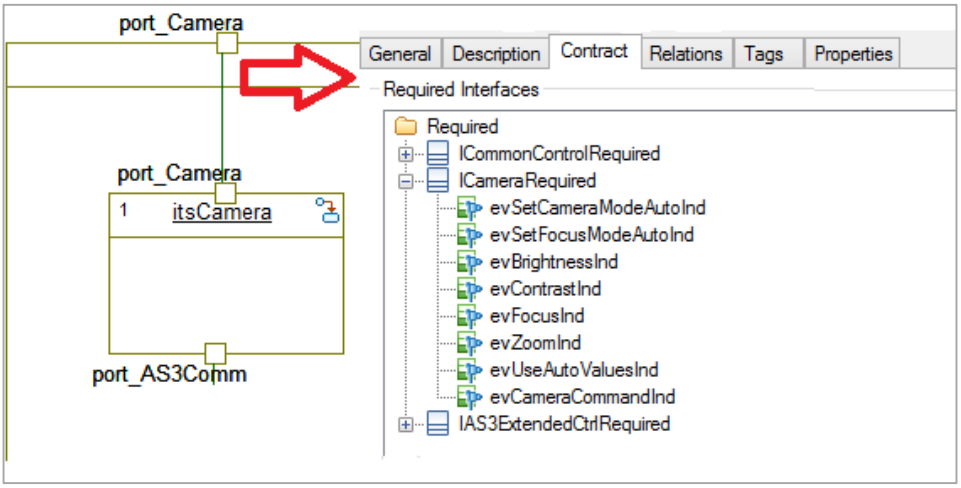

Figure 17 – Exit Data Movements

The exit data movements are captured from the sequence diagram. The exit data movement in a sequence diagram is shown in Figure 18. The exit movement has been sent by the component to the external user of the component.

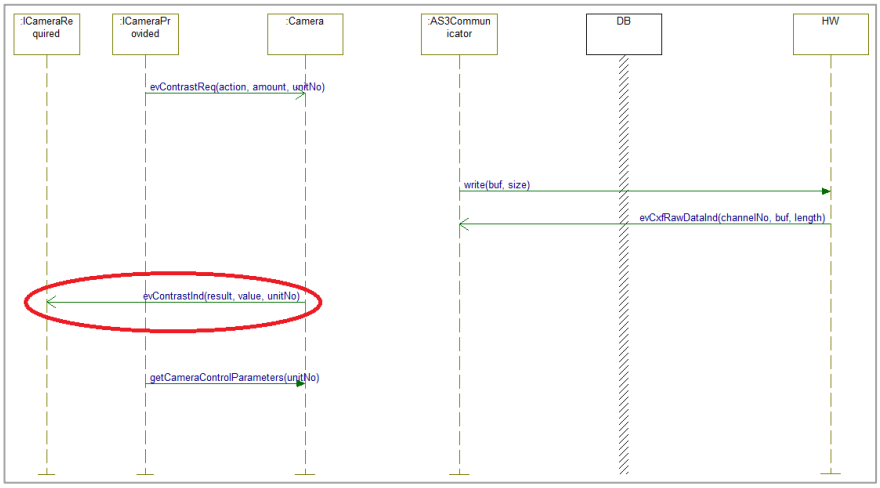

Figure 18 – Exit Data Movement in Sequence Diagram.

In order to check for consistency the sequence diagram element that is counted as an exit is controlled from the interfaces of the component located in the non-behavioral ports of the composite structure diagram.

3.1.4.3. Read Data Movement

When a functional process is in progress the read data movement types are referred from the database. Read data movement types are extracted from the sequence diagrams. The arrow going out an instance named DB (Database) in the sequence diagram is accepted as a read data movement. The read data movement is shown in Figure 19.

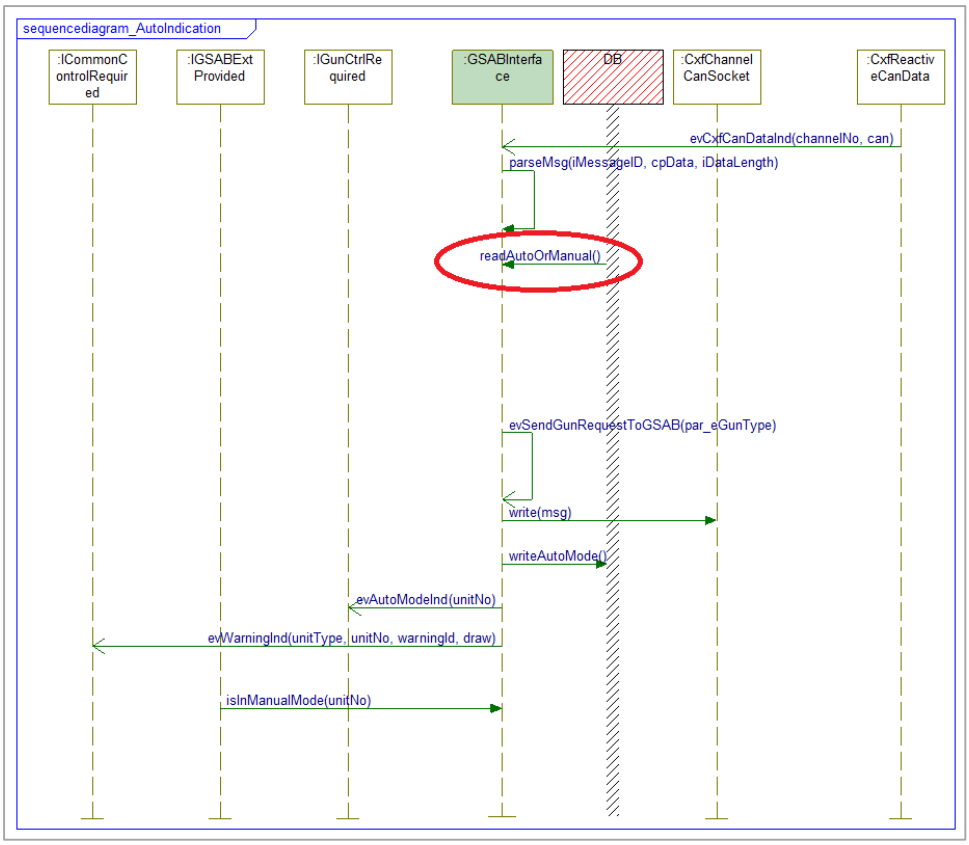

Figure 19 - Read Data Movement in a Sequence Diagram

3.1.4.4. Write Data Movement

When a functional process is in progress the write data movement types are updated in the database. Write data movement types are extracted from the sequence diagrams. The arrow going in an instance named DB (Database) in the sequence diagram is accepted as a write data movement. The write data movement is shown in Figure 20 below.

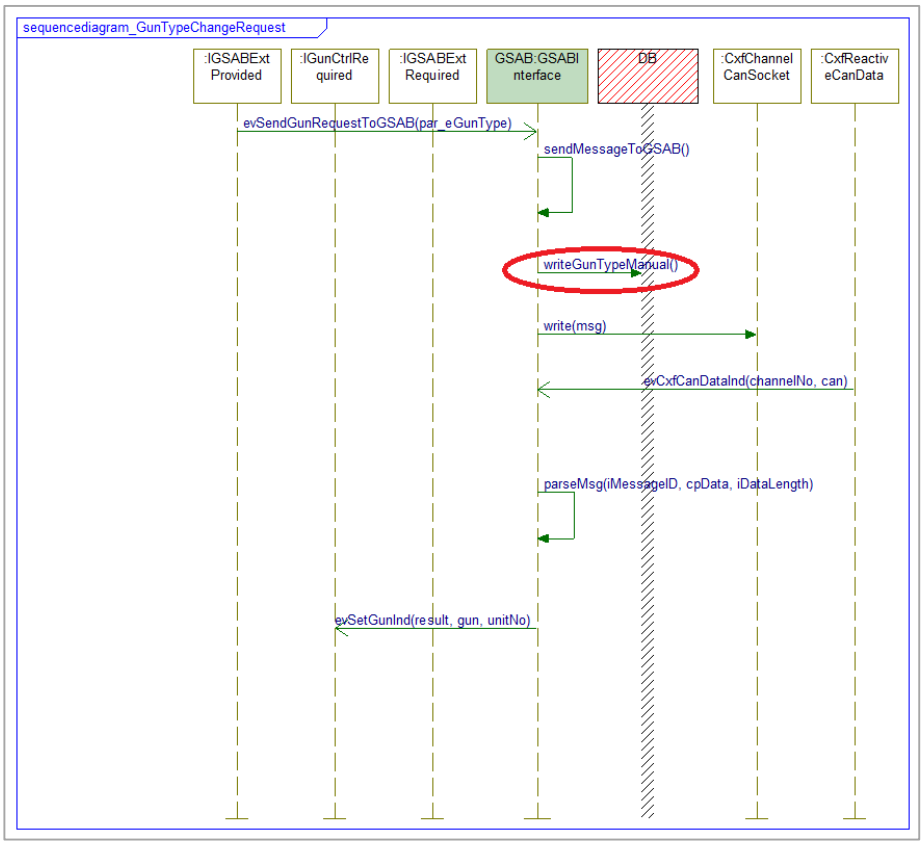

Figure 20 - Write Data Movement in a Sequence Diagram

3.1.5. Functional Process

The set of data movements exchanged between the functional user and the software to be measured to complete a task is a Functional Process. A functional process is shown in Figure 21. The functional processes are the sequence diagrams in the UML domain.

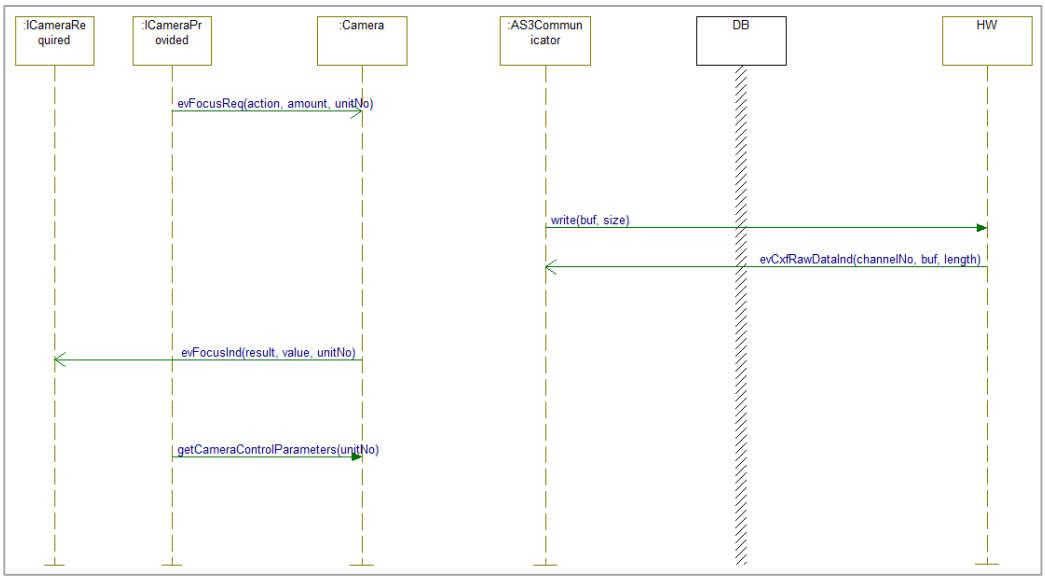

Figure 21 - Functional Process

A sequence diagram is drawn for each action defined in the Interface Design Document (IDD) of the component.

3.1.6. Data Groups

The data group is identified as the data moved by the data movement type (COSMIC Measurement Manual, 2014). The data movement types extracted from the sequence diagrams are events or attributes that have arguments. The arguments they have can be determined as the Data Groups.

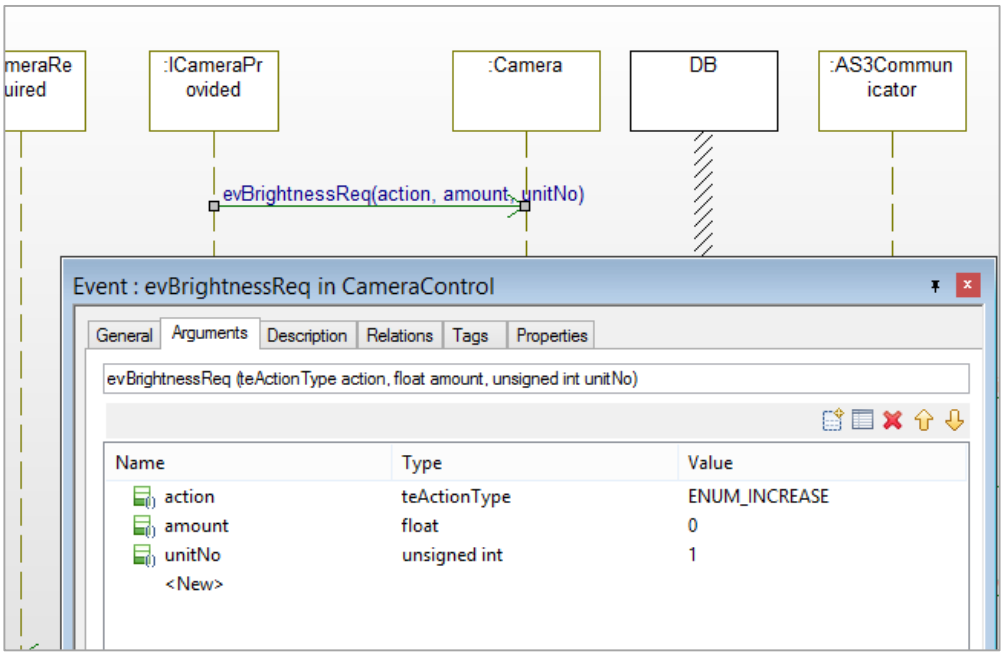

Figure 22 - Data Groups in UML Diagram

Data Groups identified in the sequence diagram are shown in Figure 22.

3.2. PL FSM TOOL

In the light of the proposed mapping, software was developed in JAVA by using NetBeans integrated development environment in order to measure the functional software size of the selected components. The NetBeans development environment is illustrated in Figure 23.

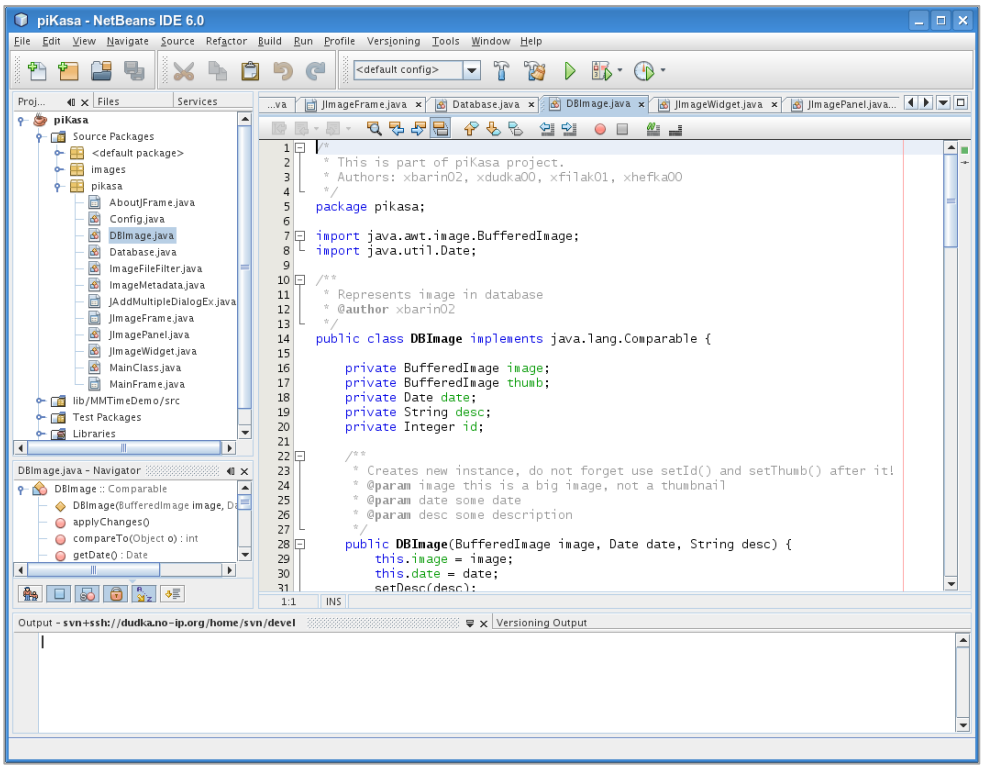

Figure 23 – NetBeans Development Environment

The software is a plug-in to the IBM Rational Rhapsody UML which explores the selected component's UML diagrams and extracts the needed information to automate the COSMIC size measurement method. The developed software uses the sequence diagram and the component diagram to extract the required information for COSMIC function point measurement. The details of the plug-in are given below.

By clicking on the selected component, software boundary is chosen as the current component boundary. The plug-in is executed after pushing AutoXMLGenerator button. The function points are shown as an output of the selected component and a document is created that describes the functional processes and data movements. The detailed results and comments for each component will be stated in chapter 5. Running the plug in is illustrated in Figure 24.

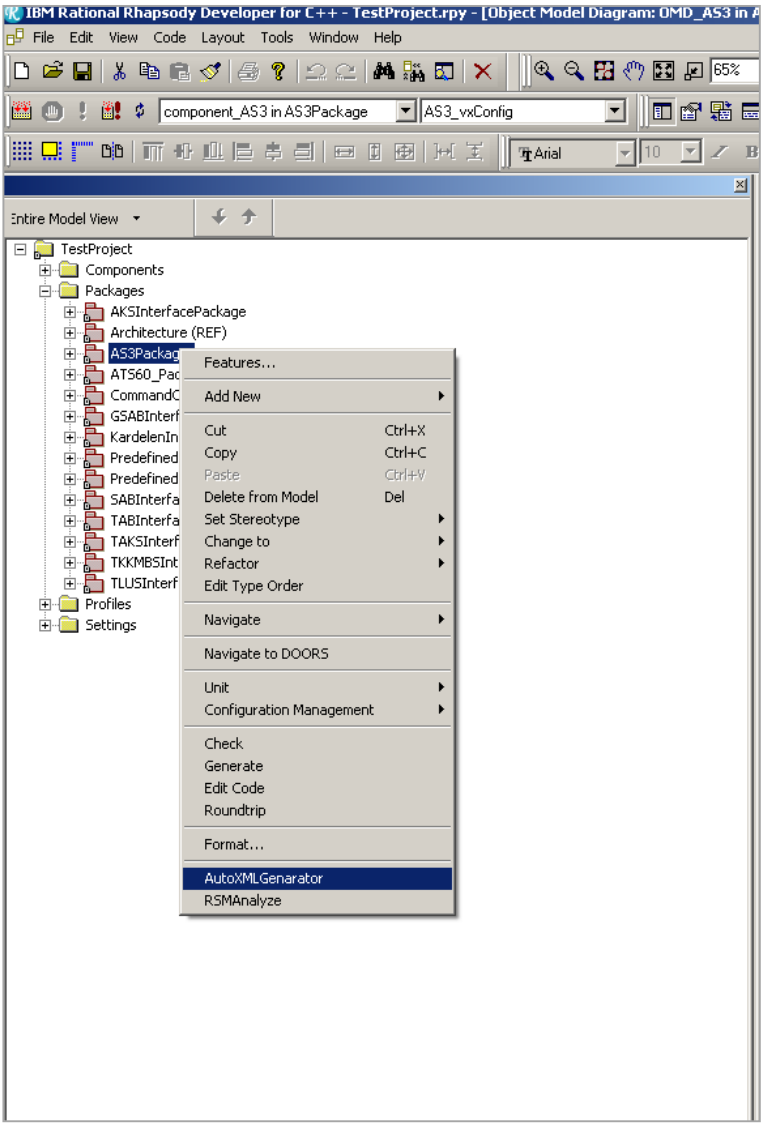

Figure 24 – Plug-in Usage in IBM Rational Rhapsody

The algorithm of the PL FSM tool is as follows:

- Search for the software boundary
- Search for the non-behavioral ports in the software boundary
- Find the number of events in the required interface of the non- behavioral port that occurs in the sequence diagram and count them as the Exit (E) data movements.
- Find the number of operations in the provided interface of the non-behavioral port that occurs in the sequence diagram and count them as the Exit (E) data movements.
- Find the number of sequence diagram elements (arrows) directed from data base instance to the software boundary which is counted as the number of the read data movements.

- Find the number of sequence diagram elements (arrows) directed from software boundary to the data base instance which is counted as the number of the write data movements.
- Sum up the calculated number of data movements to find the COSMIC function points in the functional process.
- Apply the steps recursively until all the functional processes are checked for the software boundary.
- Sum up all the COSMIC function points in every functional processes to find out the final COSMIC function points in the software boundary.

The PL FSM tool developed for the automated COSMIC FSM measurement also provides a report document that gives details about the software boundary, functional processes and data movements. Backward traceability is available with the automation tool, when a new data movement is added or deleted from the functional process there is no need to redraw the existing diagrams.

CHAPTER IV

EMPIRICAL STUDIES

All the empirical results obtained from this study are presented under this chapter. Initially, the exploratory study researched for understanding the correlation between the interface elements and functional software size is detailed. Secondly, survey conducted to get the opinions of experienced software engineers are presented under subsection 4.2. Finally, the case study investigated to validate the mapping is described.

4.1. EXPLORATORY CASE STUDY

An exploratory case study aims to have a clear view of the problem and to determine the research questions or goals (Yin, 2003). In the early stages of this research the problem definition was not clear, in order to clarify the problem and the research goals an exploratory case study is investigated details of which is given below.

This exploratory study is investigated to have an idea about the correlation between the number of the elements in a component's interface and the functional size measured manually of that component. An estimation function of COSMIC functional size is investigated by counting the number of elements in the interfaces of a component.

4.1.1. Exploratory Case Study Environment

This research has been conducted in ASELSAN which is Turkey's leading defense industry company. ASELSAN was founded by Land Naval and Air Forces Foundations in 1970 with the donations of the Turkish people in order to cover Turkey's military defense needs through national means. The company's basic strategy is to develop unique products and systems by making use of critical technologies.

ASELSAN operates in four divisions which are:

- Communications and Information Technologies,

- Defense Systems Technologies,Microelectronics, Guidance & Electro-Optics,
- Radar, Electronic Warfare and Intelligence Systems

The study is investigated in Software Engineering Department which is located in Defense Systems Technologies division.

Increasing the percentage of reused components is aimed at Software Engineering Department in order to respond to the customer requests rapidly. A component based software product line is designed to accomplish this goal. The component based software product line is structured by the interface-based design architecture. In this product line there are components which are developed by the embedded software engineers with an IBM UML tool named IBM Rational Rhapsody.

IBM Rational Rhapsody is based on the Unified Modeling Language which helps the embedded software developers to have a visual development environment. It helps the software developers to improve their productivity throughout the embedded software development lifecycle. For visualizing the software development process Rational Rhapsody has UML diagrams. In Figure 25 a sample screen of IBM Rational Rhapsody for software development is illustrated. As seen in Figure 25 there are several UML diagrams in IBM Rational Rhapsody. Some of the UML diagrams in IBM Rational Rhapsody directly affect the developed software where some of them are just for having a better view of the software developed.

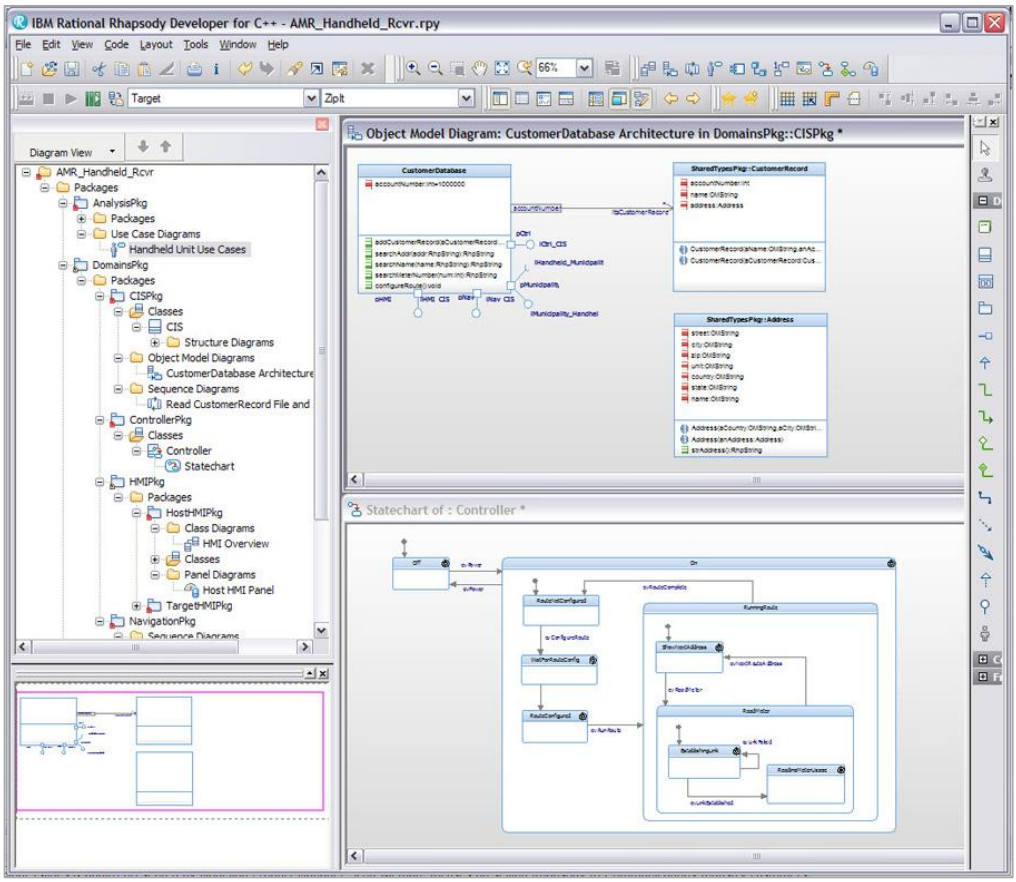

Figure 25 - IBM Rational Rhapsody

The components developed by the software engineers are then tested by the Software Test Engineering department before placing them in to the product line to serve for all the projects which need that specific component. The components are kept in a configuration management tool developed by IBM which is called IBM Rational ClearCase. IBM Rational ClearCase is a software configuration management solution that helps the developer to keep track of the version of the software, to manage the workspace and to work on the same code at the same time with a college. The product line versioned at the Rational ClearCase configuration management tool is shown in Figure 26.

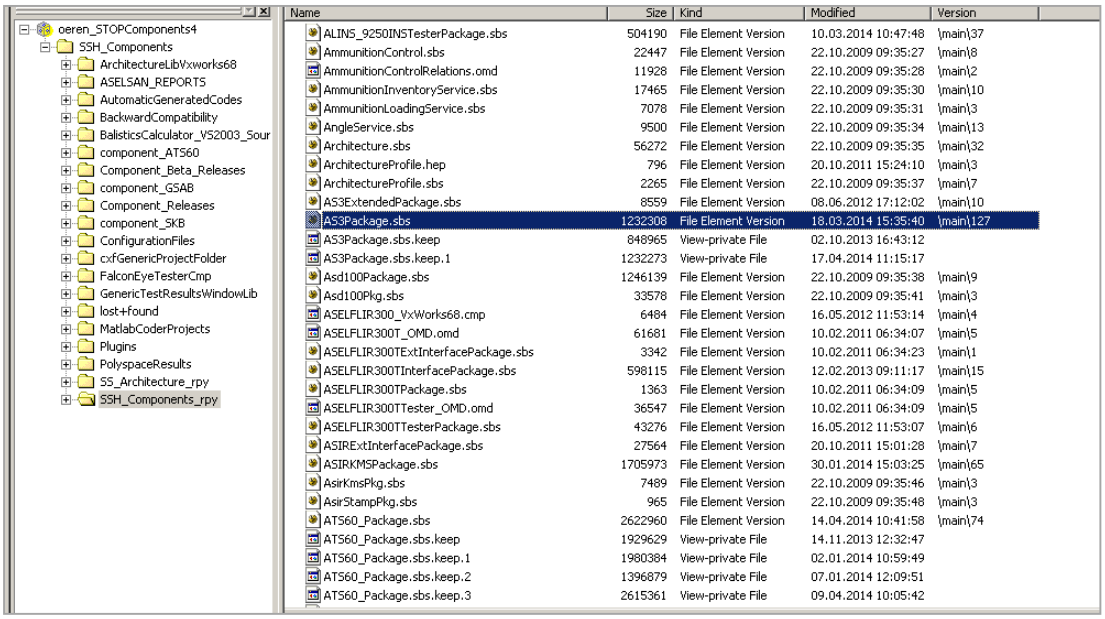

Figure 26 – Product Line

4.1.2. Exploratory Case Study Data Collection

A JAVA plug-in is developed in NetBeans development environment which counts the number of the elements in a component's ports. The ports include the interfaces between the component and the functional users of the component. The plug-in checks the non-behavioral ports located in the software boundary by means of the functions events and their arguments.

After finalizing the data collection step, the automated size estimation results are compared with the manually obtained results from the COSMIC FSM method for each product line component. The results are analyzed in SPSS. A multiple linear regression is made with the dependent variables number of operations, number of events and number of arguments and with the dependent variable function point. In chapter 4.1.3, information of missing value, outliers, and normality is detailed.

In the exploratory study 17 components are handled in the software product line. The product line components were selected for the case study from the system environment layer in the product line. This layer was taken into consideration due to its various types of components. The number of the components included in the exploratory study and their brief descriptions are given in Table 3. Due to the privacy issues, the names of the components were kept confidential.

Table 3 - Selected Component Details in Exploratory Study

Name of the Component	Description of the Component
Component_1	The component is designed to communicate with the Fire Control System Unit via tcp/ip.
Component_2	The component communicates with software via tcp/ip. Command control software provide target information to the component.
Component_3	The component communicates via tcp/ip and provides target measurement and direction for external software. It is used for surveillance.
Component_4	The component communicates via tcp/ip with external software which keeps track of the system position and status.
Component_5	The component is designed to communicate with the hardware named system commander unit which has buttons and switches on it. User interaction with the system is provided by this hardware. The component communicates with the hardware via a CAN channel.
Component_6	The component communicates via a serial channel with a hardware which is a tank laser indication system and informs the soldiers in the tank if there is a laser pointed to the tank.
Component_7	This component is designed to communicate with external software via a serial channel. The external software provides target to the component and has several modes with specific user rights.
Component_8	The component is a camera component which includes two types of camera and a

	laser to detect the distance of the target. One of the cameras is a TV camera used in clear weather conditions and the other camera is a thermal camera which provides a vision of the target by the heat difference with the environment and the target.
Component_9	This component is an enhanced version of Component_8. Its thermal camera has cooling advantage which helps the camera to have a better vision of the target in night vision mode.
Component_10	The component is designed to communicate with the hardware named system commander unit which has buttons and switches on it. User interaction with the system is provided by this hardware. The component communicates with the hardware via a CAN channel.
Component_11	The component is designed to communicate with the hardware by a CAN protocol and is used to power the units and carry out the gun processes such as firing, gun arm and safe etc.
Component_12	The component is designed to power the units and communicates with the hardware via a CAN channel.
Component_13	The component controls the power for the units and communicates with the hardware via a CAN channel.
Component_14	The component is meteorological sensor software which communicates with the hardware via serial channel.
Component_15	The component is designed to communicate with a muzzle velocity measurement sensor via serial channel.

Component_16	The component is designed to control the position of the servo motors and communicates via a serial channel.
Component_17	The component communicates with an acoustic sensor. The acoustic sensor detects the position of the threat and provides the coordinates of the threat to the component.

4.1.3. Exploratory Case Study Data Analysis

After the components were selected, the functional sizes of each component were calculated manually. The column FP in Table 4 shows the calculated FP for each of the components. The number of elements calculated with the plug-in is shown at the #Operation, #Event columns and the total numbers of all elements (#Operations + #Events) are shown in Table 4.

Table 4 - Component Statistics in Exploratory Study

Name	FP	#Element	#Operations	#Events
Component_1	16	6	1	5
Component_2	40	22	14	8
Component_3	77	30	12	18
Component_4	20	10	5	5
Component_5	35	54	21	33
Component_6	33	28	11	17
Component_7	154	77	50	27
Component_8	154	136	33	103
Component_9	215	156	32	124
Component_10	18	35	14	21
Component_11	115	96	30	66
Component_12	33	32	9	23
Component_13	48	35	15	20
Component_14	57	32	17	15

Component_15	74	45	12	33
Component_16	52	34	10	24
Component_17	42	29	14	15

When the FP and the total element number in the boundary of the component are calculated a correlation is investigated between the FP and the element number. The relation between the FP and the interface elements is shown in Figure 27. Interface element number is the sum of the number of events and operations in the interfaces of the component. It can be seen in the graph the more elements to communicate with the external components are the more is the function point calculated with the COSMIC FSM method.

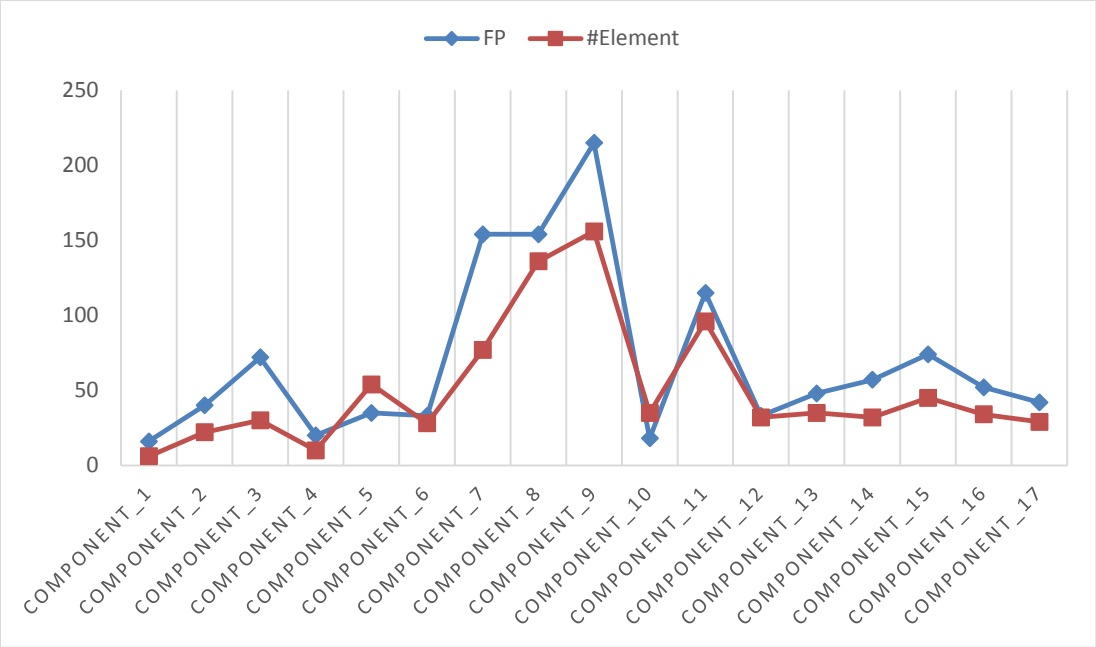

Figure 27 – FP and Elements Correlation

The correlations of the independent variables number of operations and number of events with the dependent variable function points are 0,825 and 0,860 respectively. The correlations are shown in Table 5.

Table 5 - Correlations between FP and Independent Variables

		FP	#Operations	#Events
Pearson Correlation	FP	1,000	,825	,860
	#Operations	,825	1,000	,611
	#Events	,860	,611	1,000

The coefficients of the estimate are shown in Table 6. The independent variables are statistically significant (Sig. < .05) as shown in Table 6. The estimated function points (EFP) with the independent variables Number of Operations (NOP) and Number of Events (NOE) model is as follows:

$$EFP = -1{,}355 + 2{,}234 * NOP + 0{,}962 * NOE$$

Table 6 - Coefficients Table

		Unstandardized Coefficients		Standardized Coefficients		
Model		B	Std. Error	Beta	t	Sig.
1	(Constant)	-1,355	9,120		-,149	,884
	#Operations	2,234	,541	,478	4,131	,001
	#Events	,962	,196	,568	4,909	,000

The model summary is shown in Table 7. The variance of the dependent variable FP is explained by the dependent variables by 88 percent since the R square is 0,882. The standard error of the estimate is 20 percent.

Table 7 - Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,939[a]	,882	,866	20,889

As can be seen in Figure 27, the components Component_5 and Component_10 are not behaving as the other components do. Their calculated FP is below the number of the interface elements. These components are user control components whose input is a single event with a data group that defines which button is pushed. All the actions are taken due to this event so these components' FP and number of elements in the ports are not correlated unlike the rest of the components analyzed. The COSMIC FP is lower than the number of elements in the interface of the non-behavioral port of these components. The outlier components are extracted from

the model to decrease the standard error of the estimate. FP and #Element correlation without the outliers is shown in Figure 28.

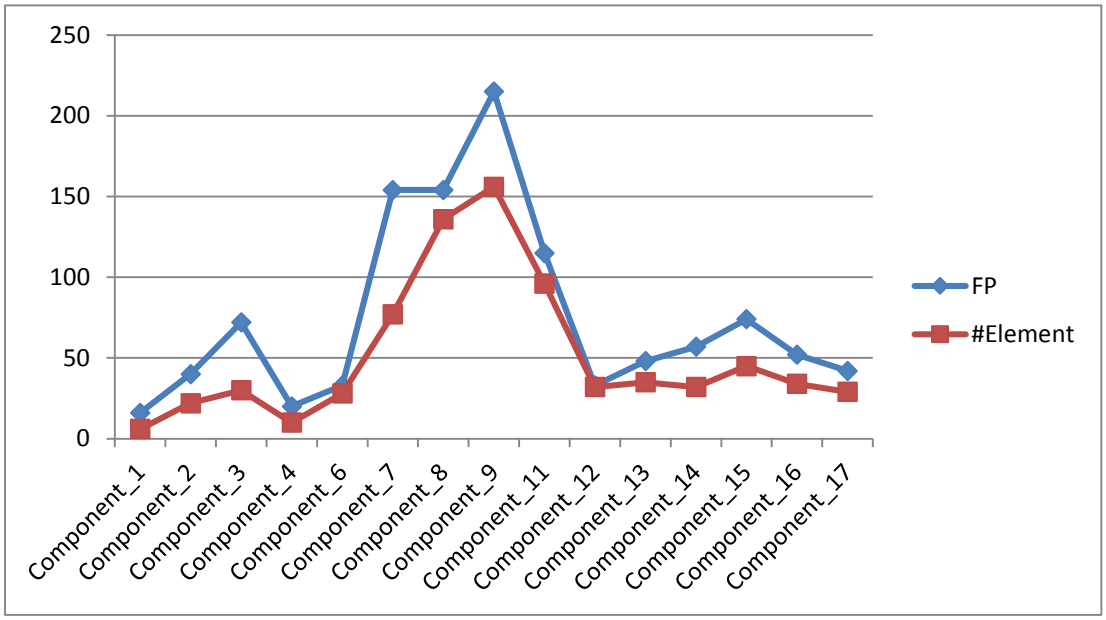

Figure 28 – FP and Elements Correlation without Outliers

Functional size and the element number in the non-behavioral ports of the component shown in Figure 28 are formulized with the multiple linear regression models. The data collected is analyzed with the SPSS. The results obtained illustrate the correlation of the independents (#Operations, #Events) with the dependent variable (FP) as shown in Table 8. The Pearson correlation between the independent variable number of operations and the dependent variable FP is 0,859 where the correlation between the independent variable number of events and the dependent variable is 0,879. The Pearson correlation between the independent variable and the dependent variables is a strong positive correlation that indicates the more elements in the interface of a component causes more functional size.

Table 8 - Correlations without Outliers

		FP	#Operations	#Events
Pearson Correlation	FP	1,000	,859	,879
	#Operations	,859	1,000	,609
	#Events	,879	,609	1,000

The P-value is calculated between these variables is lower than 0.05 which shows that the result is statistically significant. The significance is shown in Table 9.

Table 9 - Coefficients without Outliers

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	Beta	t	Sig.
1	(Constant)	3,588	6,960		,516	,616
	#Operations	2,313	,404	,514	5,725	,000
	#Events	,921	,146	,566	6,297	,000

The estimation model without the outliers calculated according to the MLR analysis is as follows.

$$EFP = 3,588 + 2,313 * NOP + 0,921 * NOE$$

This equation is an estimator of the COSMIC FP in Weapon Systems and Modernizations Team at ASELSAN with an error rate of %15 shown in Table 10. The equation coefficients may vary in other product lines which are designed by the interface-based design method. The estimation model summary is shown in Table 10.

Table 10 - Model Summary without Outliers

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	,969[a]	,939	,929	15,542

The "R" column represents the value of R, the multiple correlation coefficients. R can be considered to be one measure of the quality of the prediction of the dependent variable; FP. A value of 0.969 indicates a good level of prediction. The R Square (also called the coefficient of determination), which is the proportion of variance in the dependent variable that can be explained by the independent

variables (technically, it is the proportion of variation accounted for by the regression model above and beyond the mean model). 93.9 percent of the variability of dependent variable is explained by the independent variables.

ANOVA analysis is shown in Table 11.

Table 11 - ANOVA Analysis without Outliers

Model		Sum of Squares	df	Mean Square	F	Sig.
1	Regression	44716,771	2	22358,385	92,563	,000
	Residual	2898,562	12	241,547		
	Total	47615,333	14			

The overall regression model's fit for the data is tested in the F-ratio in Table 11. The table shows that the independent variables statistically significantly predict the dependent variable, $F(2, 12) = 91.691$, $p < .001$ value indicates that the regression model is a good fit of the data).

The calculated function point and the estimated function point for each of the components are shown in Table 12.

Table 12 - FP and Estimated FP Comparison

Component Number	**FP**	**Estimated FP**	**%Error**
Component_1	16	11	52
Component_2	40	43	8
Component_3	77	48	61
Component_4	20	20	1
Component_5	33	45	26

Component_6	154	144	7
Component_7	154	175	12
Component_8	215	192	12
Component_9	115	134	14
Component_10	33	46	28
Component_11	48	57	15
Component_12	57	57	0
Component_13	74	62	20
Component_14	52	49	7
Component_15	42	50	16

The difference between the real function point and the estimated function point is illustrated in Figure 29.

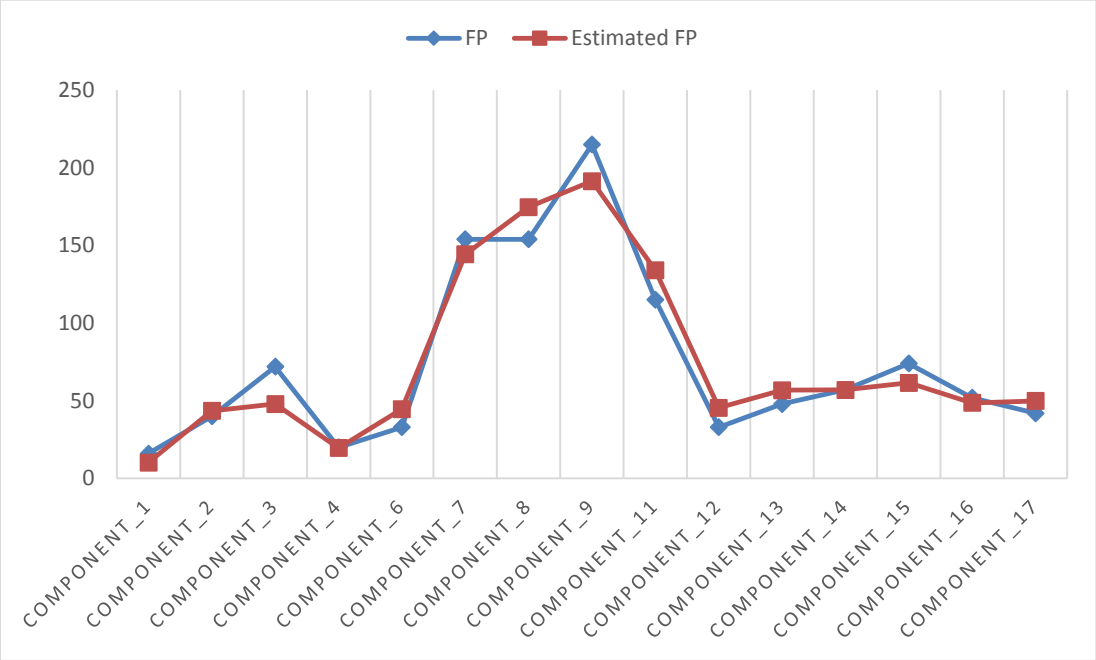

Figure 29 – FP and Estimated FP Comparison

As seen in Figure 29 the estimated values and the calculated function points are fairly close when the time gained is taken in to consideration.

4.1.4. Validity Threats for the Exploratory Study

There are several validity threats to the design of this exploratory study. The selection of the components is limited with a product line of a single software development company. Since the case study is investigating the estimation function only for the product line it is carried out the estimation results will not be similar to other product lines. The number of the components should also be higher to have more reliable results.

Another threat to the data collection is the manual measurement of the product line elements are done by the author of this study. The manual measurement results may be defected itself.

The case study is investigated in a product line that is structured in accordance with the interface-based design method which is critical in this exploratory case study. In product lines that are not designed with this method the estimation will not be valid.

4.2. SURVEY

At the beginning of the study a survey is conducted to get feedbacks from experienced embedded software developers about the size measurement process in a UML

environment structured in accordance with the interface-based design architecture. The survey questions have been replied by 13 participants who have UML and software product line experience. At the beginning of the survey, the participants were given a brief presentation about COSMIC size measurement method. The purpose of the survey is also described carefully to the participants at the beginning of the survey. The survey consisted of 10 questions which are prepared by the author of this study. They were asked which UML diagrams should be used to capture the COSMIC elements to automate the size measurement process. The questions were carefully prepared for not to misguide the respondents. The descriptive results of the survey are given in details under this chapter. Conducted survey is given in Appendix A section.

The participants were asked 9 questions in various types. The first two of the questions in the survey were in the type of demographic questions which are used to identify characteristics such as number and experience. One of the questions in the survey was in the type of open-ended question in which the participants' opinions about the UML diagrams and COSMIC concept is asked. Another type of survey questions that took part in the survey was a semantic differential scale type of question in which the attitude of the participants' about size measurement is investigated. The participants were asked to rate how important they think size measurement was for them. The rest of the survey questions were in the type of dichotomous question in which a yes or no reply is required.

All types of the questions in the survey were evaluated in their specific evaluation methodologies. In the open-ended question types the key words defined earlier were counted. The demographic types of questions were used for taking the work experience of the participant. The semantic scale question type is evaluated by finding the average result.

4.2.1. Participants

When selecting the participants for the study, it has been considered to select the participants from the software engineers who have UML experience, common knowledge about software product lines and interface-based design architecture. 13 participants, including 3 women, 10 men attended to the study. Participants were selected among the people who have been graduated from the faculty of engineering and have at least two years of work experience.

The work experience in terms of years of the 13 participants was given in Table 13.

Table 13 - Participants' Specifications

# Participant	Experience(year)
P 01	15
P 02	13
P 03	7

P 04	13
P 05	12
P 06	3
P 07	2
P 08	3
P 09	6
P 10	8
P 11	8
P 12	13
P 13	9

The distribution of the years of software engineering experience of the participants is given in Figure 30. The least experienced software engineer has 2 years of work experience and the most experienced participant has 15 years of work experience. The average work experience of the 13 participants is 8.5 years.

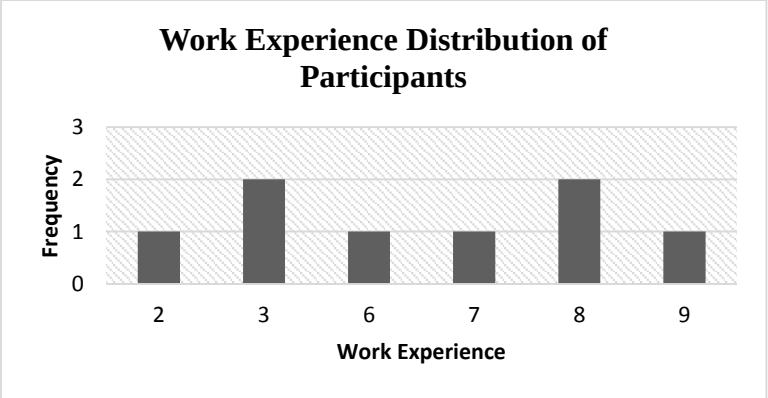

Figure 30 –Experience Distribution of the Participants

4.2.2. Survey Results

The first two questions were demographic questions which were for learning the participant profile. The average work experience level of the participants is 8.5 in terms of years which shows they are experienced in the related fields SPL, UML and IbD. 76 percent of the participants are experienced in these fields more than 5 years.

The participants were asked, how important software size measurement in the earlier phases of a project was with a 5 scale measurement where 1 is not important and 5 is extremely important. Most of the participants remarked that size

measurement in the design phase is important. The average scale has been calculated as 3.6 over 5.

The participants were asked if they have ever measure software size to have a view of their size measurement experience. 70 percent of the participants stated that they have measured software size before. However, none of the participants measured software size with COSMIC FSM method.

All of the participants stated that they would be encouraged to measure software size if the process would be automated with the UML diagrams used in the design phase. This result can be interpreted that the software developers do not measure software size because the process is hard and time taking.

92 percent of the participants declared that software boundary can be obtained from the sequence and composite structure diagrams.

The participants agreed on that functional user can be obtained from use case diagrams with a majority of 61 percent.

84 percent of the participants state that sequence and composite structure diagrams are sufficient for obtaining read, write, entry exit and triggering events.

Participants also state that it is possible to measure software size measurement with the UML diagrams used in the design phase with a majority of 84 percent.

Survey indicates that the experienced users who have SPL, IbD and UML experience think it is possible to automate the software size measurement by COSMIC FSM method using UML diagrams. The most suggested diagram for obtaining the COSMIC elements is the sequence diagram. Since the sequence diagrams and composite structure diagrams are frequently used diagrams in the design phase of a component, automating the software size measurement by using these diagrams would be efficient.

4.3. MANUAL AND AUTOMATED COSMIC FSM COMPARISON CASE STUDY

In this case study, the manual results provided by the COSMIC FSM expert has been compared with the automated results obtained with the mapping described in Chapter 4. First, the company that the case study is investigated in is briefly described. Moreover the case study implementation is explained and results of the case study are discussed. Finally, the validity threats are detailed.

4.3.1. Case Study Environment

The environment of this Case study is given at Chapter 4.1.1.

4.3.2. Case Study Data Collection

After the respondents' opinions were evaluated the components in the product line of Software Engineering Department is used to collect the data needed. 5 components were selected from the product line due to their types in order to measure the functional sizes of the components manually by using COSMIC FSM method. The reason why these 5 components were selected is that their Software Requirements Specification (SRS) and Interface Design Documents (IDD) were complete.

The product line components were selected for the study from the system environment layer in the product line. This layer was taken into consideration due to its various types of components. For instance, there are several types of camera components, power control components and user control components. The number of the components included in the study and their brief descriptions are given in Table 14. The names of the components were not given due to confidentiality constraints.

Table 14 - Component Descriptions

Name of the Component	Description of the Component
Component_18	A camera component that communicates with hardware. Hardware has two types of camera which are TV and thermal and a laser to measure the distance of the target.
Component_19	A system and weapon control software that communicates with hardware. The hardware controls the gun and the system power by means of user interaction.
Component_20	Meteorological sensor software that communicate with hardware. The hardware provides temperature, pressure, humidity and wind information to the user.
Component_21	Acoustic sensor software that communicate with hardware. The hardware provides the target information to the user.
Component_22	A system control unit software that communicates with hardware. The hardware

	is a keyboard that has buttons, switches and commutators on it.

When the survey results were analyzed and the product line components were selected, a COSMIC concept and UML conceptual elements were mapped in the light of the survey results and related researches. The mapping was given in Chapter 3.

In the light of this mapping, PL FSM tool described in Chapter 3 is developed to automatically obtain the functional software size of the selected components.

Manual measurement results of the components were needed to validate the mapping and the measurement tool. The functional software size of the components was measured manually by a certified COSMIC measurement expert who has 5 years of experience in this field. The measurement expert is certified by the COSMIC FSM consortium. SRS and the ICD documents were used by the COSMIC expert to measure the functional software size.

The duration of the measurement process for each component was written down by the expert. The manual measurement results were given in Appendix B section.

Finally, the manual measurement results provided by the certified COSMIC FSM expert were compared with the automated results obtained by the automation tool. The automated and manual measurement comparison was made by the means of size measurement duration and function points calculated. The comparison results were given in Chapter 5.

4.3.3. Case Study Data Analysis

After the survey results were analyzed, it was decided that the UML diagrams for capturing the COSMIC elements functional user, software boundary, triggering event, read, write, entry and exit would be the sequence and composite structure diagrams.

The selected components were measured by the plug-in developed. The function points and measurement duration are recorded. The measurement duration and calculated FP by the automated method are shown in Table 15.

Table 15 - Automated Measurement Results

Component Number	Calculated FP	Measurement Duration
Component_18	174 FP	40 min

Component_19	104 FP	35 min
Component_20	56 FP	25 min
Component_21	36 FP	17 min
Component_22	66 FP	15 min

The time needed to draw the sequence diagram for automated measurement is taken into account and this is the reason why the automated measurement time is longer than expected.

The same components are then manually measured by a certified COSMIC FSM expert in order to compare the results with the automated method and validate the COSMIC – UML mapping. The manual measurement results calculated by the certified COSMIC FSM expert are presented in Table 16.

Table 16 - Manual Measurement Results

Component Number	**Calculated FP**	**Measurement Duration**
Component_18	129 FP	60 min
Component_19	99 FP	45 min
Component_20	41 FP	35 min
Component_21	30 FP	30 min
Component_22	63 FP	25 min

The average time gained by the automated measurement is 33 percent and the average difference between the automated and manual measurement results is 14 percent. The reason of the measurement results difference will be detailed in this chapter. The differences between the manual measurement and the automated measurement are given in Table 17.

Table 17 - Comparison of Manual and Automated Measurement Results

Component Number	Time Difference	FP Difference
Component_18	+ 33%	25%
Component_19	+ 22%	4%
Component_20	+ 29%	26%
Component_21	+ 43%	20%
Component_22	+ 40%	1%
Average	**33%**	**15%**

Component_18 is selected in order to give best practice details and to describe the manual and automated measurement results of the components. The component diagram of Component_18 is illustrated in Figure 31.

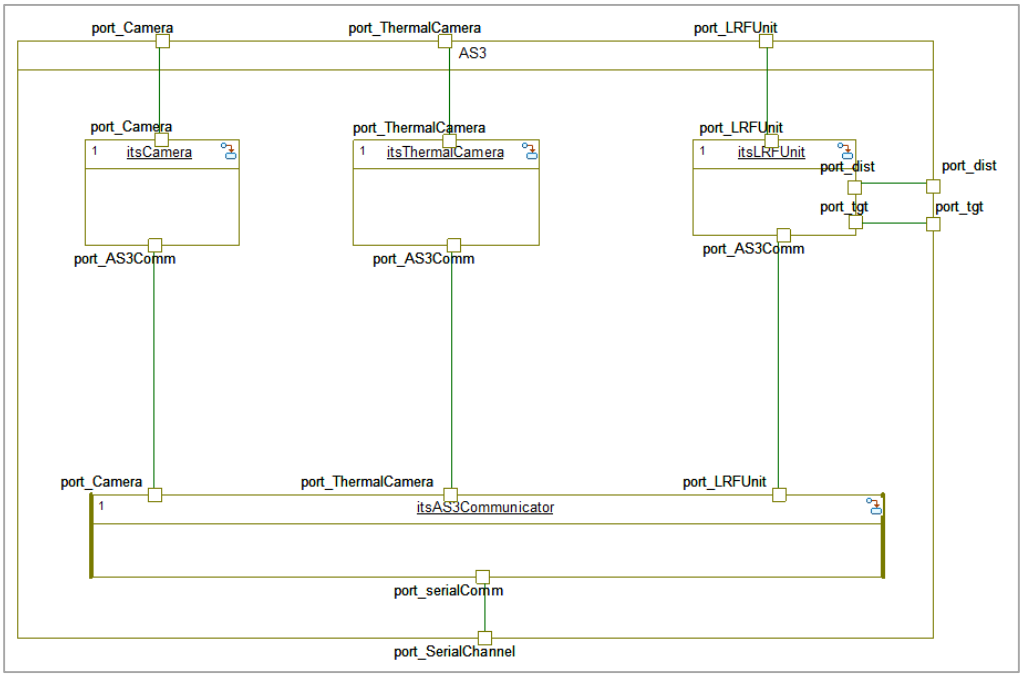

Figure 31 - Composite Structure Diagram of Component_18

The details of Component_18 can be found at Chapter 3. As seen in Figure 31 there is also a communication class in the object model diagram to provide communication of the three classes. The ports that are located in the boundary of the composite structure diagram include the interfaces to communicate with the external users of the component. The users of the component use these ports to give instructions and get the feedbacks of the actions regardless of what is done inside the software boundary.

The results obtained by the automated method for Component_18 is given in Table 24 at Appendix C.

As can be seen from the Table 1, automated measurement has determined the functional processes from the sequence diagrams which are the scenarios of each action in a component. The triggering events are also determined by the plug-in the first element of the sequence diagram is accepted as the triggering event. The consistency check is done by the help of the component diagram for finding the data movements. Data movements and their counts are detailed in Table 24.

4.3.4. Case Study Discussion

The %26 difference between the automated measurement and the manual measurement for the Component_18 is caused by the architecture used in the product line. The expert has counted the entry data movement that is directed from the hardware to the software boundary as a single data movement. Actually, it is the right thing to do but because of thef architectural limitations an entry from the hardware consists of a number of entry and exits shown in Figure 32.

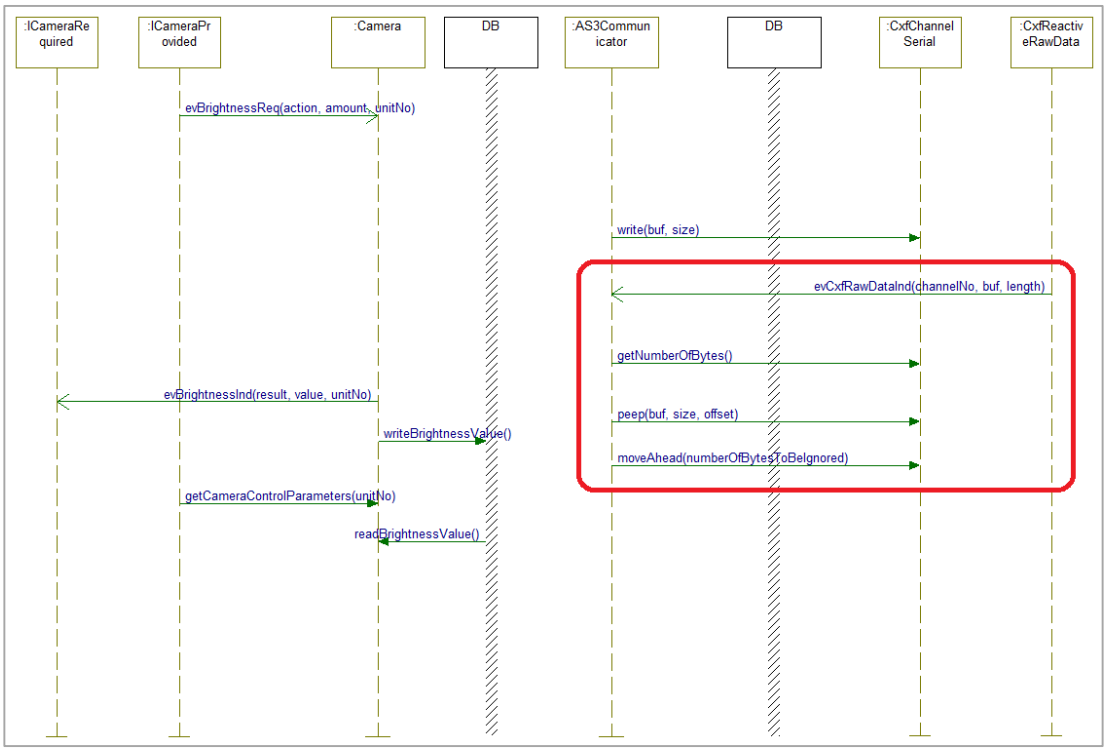

Figure 32 - Entry from Hardware via Serial Channel

In Figure 32 the data exchange between the serial channel and the component is explained. Since the expert made the measurement aware of the architectural design and the SRS document does not describe how the communication is provided via the hardware the difference in measurement results is inevitable. Component_19 and Component_22 results support this explanation. The difference between the automated and manual measurement is calculated %4 and %1 respectively. Communication with the hardware is provided via a CAN interface and according to the architectural design they exchange data with the hardware by a single data movement as seen in Figure 33.

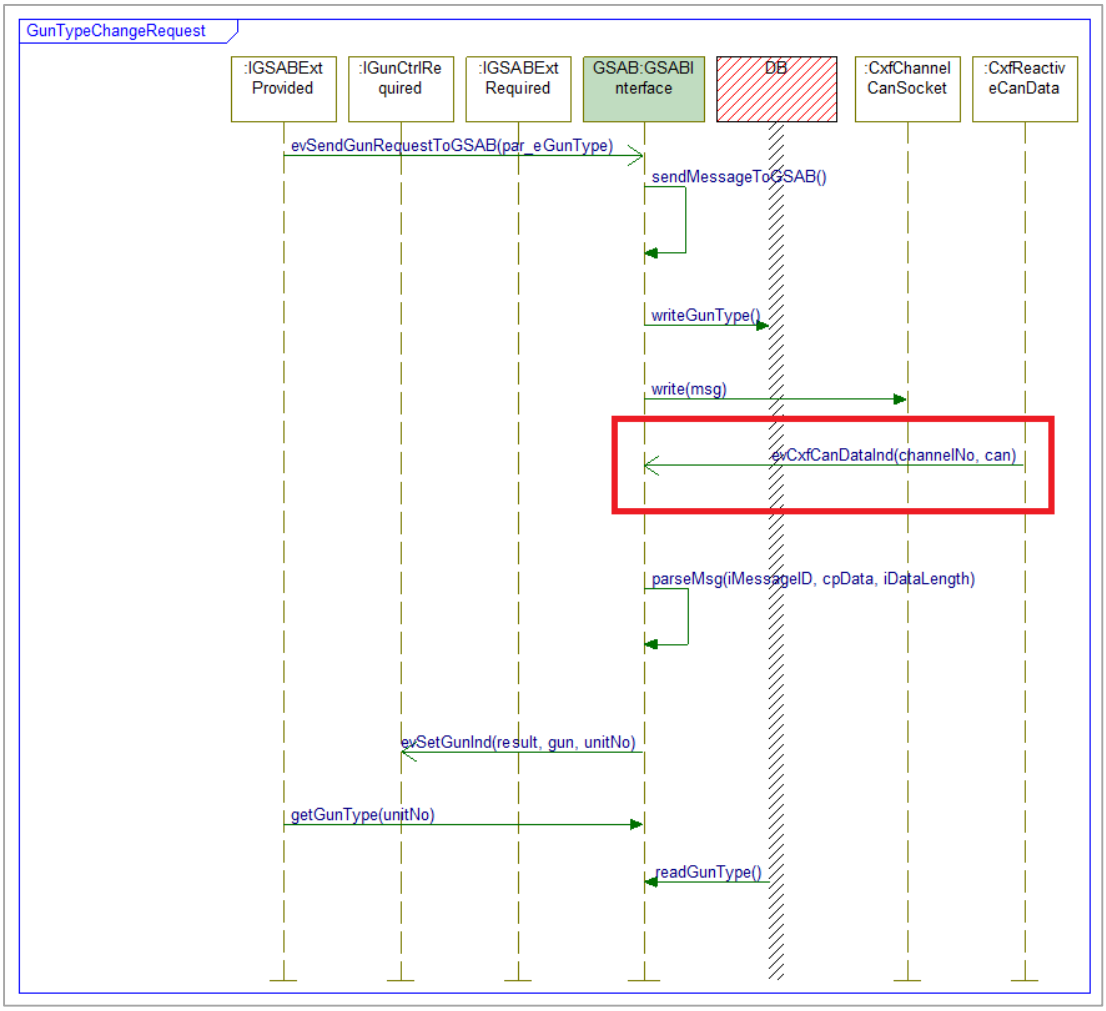

Figure 33 - Entry from Hardware via CAN Channel

The average time gained in the automated measurement compared by the manual measurement is 33 percent lower as seen in Table 17. The time gained is caused by the architectural design in the product line. Since the interfaces in the ports of the components are defined earlier and the inputs and the outputs are known by the software engineer it takes shorter time to define the entry and exit data movements. The interfaces of Component_18 is given in Figure 34 to provide better understanding of the situation.

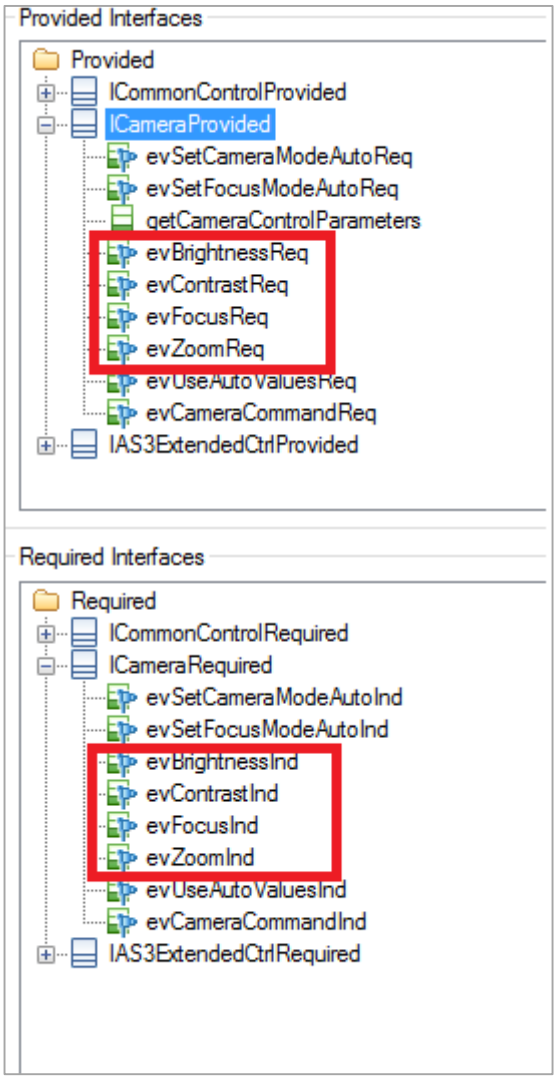

Figure 34 - Interfaces of the Component_18

As can be seen from Figure 34 the provided interface includes the evBrightnessReq, evContrastReq, evFocusReq and evZoomReq which are the entry data movements in COSMIC terms and by the measurer it is easily identified that evBrightnessInd, evContrastInd, evFocusInd and evZoomInd are the exit data movements located in the required interface. In a nutshell, interface-based design architecture provides shorter COSMIC measurement time and gives the measurer a broader view that makes sequence diagram easier to draw.

4.3.5. Validity Threats for the Case Study

There are several validity threats to the design of this study. The selection of the components is limited with a product line of a single software development company. In extending this work we should of course include more components

from a variety of product lines. The number of the components should also be higher to have more reliable results.

Another threat to the data collection from the UML diagrams for the automated measurement process is that the UML diagrams were drawn by the author of this thesis; it may include some defects itself. In order to minimize the manual measurement errors the manual measurement process was carried out by an expert who has COSMIC FSM certification and at least 5 years measurement experience.

The case study is investigated in a product line that is structured in accordance with the interface-based design method which is critical in the study. In product lines that are not designed with this method the measurement may not create similar results.

The selected components to check the mapping of UML and COSMIC elements were developed for real time embedded systems; in other domains such as MIS similar results may not be obtained.

CHAPTER V

DISCUSSION AND CONCLUSION

In this final chapter, discussion and conclusion of the research are given. Contribution of the study can be found in section 5.2. Finally, directions for further research are presented and the limitations of the study are detailed.

5.1. DISCUSSION

This study is investigated in order to reduce the human effect in functional size measurement in component based product lines by using the UML diagrams. The motivation of the study was to propose an effective automation method for functional size measurement in CBPL environment by automating the measurement process. In the light of this motivation, the following research goals are set.

- Determining the UML diagrams and diagram elements for functional size measurement in CBPL that are structured in accordance with the IbD method.
- Automating the COSMIC FSM by UML diagrams in CBPL environment.

Achievement of these research goals are discussed below.

In order to determine the UML diagrams and elements for functional size measurement in CBPL environment a survey was conducted to get the opinions of experienced software engineers. In the light of the survey results, COSMIC conceptual elements are mapped with the UML diagram elements. Composite structure diagrams and the sequence diagrams are selected for automating the COSMIC functional size measurement. Composite structure diagram is used to obtain the interfaces located in the non-behavioral ports of the component. In addition sequence diagrams are used to capture the functional processes and the data movements together with the component diagram. The functions and the events in the composite structure diagram's ports located in the software boundary are used to identify the entry and exit data movements in coordinate with the sequence diagram's directed arrows. In addition sequence diagram's instance line, system border, event and message elements are used to identify the read, write entry and exit data movements.

The size measurement processes need expertise to obtain realistic results. Human error in the calculation may cause unreliable measurement. To get the human factor out of the way this procedure is automated. In this study the size measurement procedure is applied in a limited design technique called the interface-based design method. For measuring the functional software size the elements of object and sequence diagrams are used as an input to the functional size measurement tool called PL FSM. PL FSM is a novel approach that supports organizations in effective FSM practices by providing a UML based mapping which is a defacto specification language in such environments. PL FSM makes use of the FUR information embedded in the component interfaces in order to measure the functional software size. Since the approach is fully UML based the significance of the study is high.

The functional size measurement methods have certain rules and steps defined in depth in their user manuals. The proposed PL FSM method reduces measurement duration by 33 percent. The Table 18 below illustrates the measurement durations for each component in manual and automated way.

Table 18 - Measurement Duration Comparisons

Component Number	Manual Measurement	Automated Measurement
Component_18	60 min	40 min
Component_19	45 min	35 min
Component_20	35 min	25 min
Component_21	30 min	17 min
Component_22	25 min	15 min

Since the software size measurement is carried out before the SRS document of the software has been written the design effort is also saved in the introduced automated software size measurement.

5.2. CONCLUSION

Evaluating the project in the design phase is crucial for software management however the measurement process is challenging and needs expertise. These reasons create the need to automate the measurement process.

This research was conducted with two case studies. The first case study was an exploration study that is investigated to have a clear view of the problem and to determine the research questions or goals. The second case study is investigated in order to validate the proposed mapping between the UML diagram elements and COSMIC conceptual elements.

As a result of the exploratory study, it was shown that the approximation technique researched provides close results with an acceptable error rate with the function point for each product line component calculated by the COSMIC FSM method. The number of the functions, events and their arguments in the pre-defined interfaces of the product line components which are designed via interface-based design method can be used for approximating the functional size of that component. When a new component has to be developed in the product the functional size can be known with an acceptable error rate compared to the empirical estimation with a lower effort for estimating the FP. The estimation is done before nothing has been coded by just taking the component's pre-defined interfaces into account. The tool developed in JAVA helps the user to calculate the inputs used in the approximation instantly. By applying this method the functional size estimation can be done by a more quantitative method compared to the empiric method used earlier in ASELSAN Weapon Systems and Modernizations Team. The error rate in the estimation process by using the historic data is about %25 to %30 according to the data provided by ASELSAN. The data is not shown in this document because of the privacy issues in ASELSAN. In the functional size approximation method done by using the elements in the predefined interfaces, the absolute mean error is calculated %15. Compared with the old method used in ASELSAN the error rate is lower. The effort for estimating the functional size is lower than the previous method and it is an acceptable error rate when the time gained is taken in to account.

The case studies in this study have been carried out in an embedded systems product line however the results can be generalized in other software development environments.

In this study the functional size of software components are calculated with the PL FSM approach. The functional software size can be used as effort information by converting the functional size to effort with the proposed techniques in the literature.

In this paper, measurement rules are proposed to automate the software size measurement by the frequently used UML diagrams in the light of COSMIC FSM and UML mapping given in Table 1. Automation of the measurement process is carried out in a software product line environment which is structured by the interface-based design method. The time saved in the measurement process is 33 percent calculated

by comparing the manual measurement duration and the automated measurement duration. The time saved by writing the SRS document is not included in this result. The automated measurement results and the manual measurement results have a difference by 14 percent that is arisen of the design limitations of the product line in which the study is carried out.

In conclusion, avoiding the measurement errors, obtaining objective measurement results and reducing the measurement duration is possible by automating the software size measurement in UML context.

5.3. CONTRIBUTION OF THE STUDY

Software developing companies concentrate on delivering the software as quickly as possible. In fact, these companies do not spend much time on functional size measurement and trust on their historical data in estimating the size of the software to be developed. This type of approach is not scientific and may result with high error rate in the estimation process. This study suggests an automated size measurement of COSMIC FSM by obeying the context of the method and employing frequently used UML diagrams in the design stage.

There is numerous automated functional size measurement methods proposed in the literature, however most of them picked components for their case study not from the real world. In this study, the subjects were selected from an existing product line of a software developing company.

This study contributes to the automated COSMIC functional size measurement literature by measuring the functional size employing the frequently used UML diagrams in the design stage.

Most of the studies are in the domain of MIS however there are not so many researches in the field of real time systems. This study also contributes to the literature by the real time domain research results about the automated COSMIC functional size measurement.

Furthermore, it is observed that there are not so many researches made about the size measurement in a product line environment. The study was validated by the components selected from a product line of a software developing company.

The findings of this study show that functional size can be measured in the design stage by using the UML diagrams. The effect of the architecture called the interface-based design architecture is making the functional size measurement process easier and quicker. The study encourages the software developers to allocate time with the software size measurement of the software they cope with before they have start to code. The experience needed for the COSMIC software size measurement process is also decreased as a contribution of the study.

5.4. LIMITATIONS AND FURTHER RESEARCH

There are a small number of limitations which affect the results of the study investigated. The most important limitation of the study is the environment of the study. The study is examined in a product line environment in which the components are developed in a certain design style called interface-based design method.

The study is carried out in the embedded systems product line which is also a limitation for the study.

Another limitation for the research is the type of the software components subjected. The components examined are the system environment layer components which are such gates that provide communication between the upper layer and the hardware layer. They do not contain algorithmic operations within them. The number of the components is also not sufficient to have more reliable results.

Future research should concentrate on checking the automated measurement process in Management Information Systems (MIS) to see if the results are valid for that environment. The automation process can also be taken one step further by drawing the sequence diagrams automatically by using the ICD for each component. Increasing the number of the components measured manually and automatically may give more accurate results for calculating the time gained with the automation process. The study should also be validated with multiple case studies in several product lines of different companies.

REFERENCES

Abran, A. (1999, October). COSMIC FFP 2.0: An implementation of COSMIC functional size measurement concepts. In *Proceedings of FESMA '99* (pp. 29-38). Amsterdam, Nedherlands.

Abran, A. (2010). *Software metrics and software metrology*. Toronto: John Wiley & Sons Inc.

Abran, A., & Robillard, P. N. (1996). Function points analysis: an empirical study of its measurement processes. *Software Engineering, IEEE Transactions on, 22*(12), 895-910. doi: 10.1109/32.553638

Abran, A., Desharnais, J. M., Oligny, S., St-Pierre, D., & Symons, C. (2003). *Cosmic-ffp measurement manual, version 2.0. Software Engineering Management Research Laboratory.* Montreal, Canada: Universite´ du Quebe´c a` Montre´al.

Albrecht, A. J. (1979, October). Measuring application development productivity. In *Proceedings of the Joint SHARE/GUIDE/IBM Application Development Symposium* (Vol. 10, pp. 83-92). New York, USA.

Atkinson, C., Paech, B., Reinhold, J., & Sander, T. (2001, September). Developing and applying component-based model-driven architectures in KobrA. In *Proceedings of 5 th Enterprise Distributed Object Computing Conference, Fifth IEEE International* (pp. 212-223). Seattle, USA.

Atkinson, C., Bayer, J., & Muthig, D. (2000, November). Component-based product line development: the KobrA approach. In *Proceedings of the First Conference on Software Product Lines: Experience and Research Directions* (pp. 289-309). Denver, USA.

Azzouz, S., & Abran, A. (2004, January). A proposed measurement role in the rational unified process and its implementation with ISO 19761: COSMIC-FFP.

In *Proceedings of the Software Measurement European Forum* (pp.1-12). Rome, Italy.

Bate, I., Hawkins, R., & McDermid, J. (2003, October). A contract-based approach to designing safe systems. In *Proceedings of the 8th Australian workshop on Safety critical systems and software-Volume 33* (pp. 25-36). Darlinghurst, Australia

Berg, K., Dekkers, T., & Oudshoorn, R. (2005, March). Functional size measurement applied to UML-based user requirements. In *Proceedings of the 2005 SMEF Conference* (pp.69–80). Rome, Italy.

Berardi, D., Calvanese, D., & De Giacomo, G. (2005). Reasoning on UML class diagrams. *Artificial Intelligence, 168*(1), 70-118. doi: 10.1016/j.artint.2005.05.003

Bévo, V., Lévesque, G., & Abran, A. (1999, September). Application de la methode FFP a partir d'une specification selon la notation UML: Compte rendu des premiers essais d'application et questions. In *9th International Workshop Software Measuremen*. Lac Supérieur, Canada.

Bhatt, K., Vinit, T., Patel, P., Mits, K. B., & Ujjain, D. (2012). Analysis of source lines of code (SLOC) Metric. *International Journal of Emerging Technology and Advanced Engineering, 2*(5), 150-154. Retrieved from: http://www.ijetae.com/files/Volume2Issue5/IJETAE_0512_25.pdf

Boehm, B. W., & DeMarco, T. (1997). Software risk management. *IEEE Software, 14*(3), 17-19. doi: 10.1109/MS.1997.589225

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified Modeling Language Reference Manual, (the 2nd edition).* Pearson Higher Education.

Bosch, J. (2002, August). Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Proceedings of Second Software Product Lines* (pp. 257-271). San Diego, USA.

Breivold, H. P., & Larsson, M. (2007, August). Component-based and service-oriented software engineering: Key concepts and principles. In *Software Engineering and Advanced Applications, 2007. 33rd EUROMICRO Conference on* (pp. 13-20). IEEE.

Brown, A. W. (2000). *Large-scale, component-based development* (Vol. 1).New Jersey: Prentice Hall.

Brown, A. W., & Wallnau, K. C. (1998). The current state of CBSE. *IEEE software*, *15*(5), 37-46.

Cantor, M. (1998). *Object-oriented project management with UML*. Toronto: John Wiley & Sons Inc.

Cheesman, J., & Daniels, J. (2000). *UML components: a simple process for specifying component-based software*. New Jersey: Addison-Wesley Longman Publishing.

Clauss, M. (2001, September). Generic modeling using UML extensions for variability. In *Workshop on Domain Specific Visual Languages at OOPSLA*, (pp.11-18). Tampa, USA.

Clements, P., & Northrop, L. (2001). *Software product lines: practices and patterns*. Addison-Wesley.

Common Software Measurement International Consortium. (2014). *COSMIC-measurement manual, version 4.0.*

De Alfaro, L., & Henzinger, T. A. (2005). *Interface-based design* (pp. 83-104). Netherlands: Springer.

Del Bianco, V., & Lavazza, L. (2009, June). Applying the COSMIC functional size measurement method to problem frames. In *Engineering of Complex Computer Systems, 2009 14th IEEE International Conference on* (pp. 282-290). Potsdam, Germany.

Dikel, D., Kane, D., Ornburn, S., Loftus, W., & Wilson, J. (1997). Applying software product-line architecture. *Computer*, *30*(8), 49-55. doi: 10.1109/2. 607064

Enselme, D., Florin, G., & Legond-Aubry, F. (2004). Design by contracts: Analysis of hidden de-pendencies in component based applications. *Journal of Object Technology*, *3*(4), 23-45. Retrieved from: http://www.jot.fm/issues/ issue_2004_04/article2/

Farr, L., & Nanus, B. (1964). *Factors that affect the cost of computer programming* (No. TM-1447/000/02). System Development Corp Santa Monica CA.

Fetcke, T., Abran, A., & Dumke, R. R. (2011). 2.1 A Generalized Representation for Selected Functional Size Measurement Methods. *COSMIC Function Points: Theory and Advanced Practices*, 89.

Frakes, W. B., & Kang, K. (2005). Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, *31*(7), 529-536. doi: 10.1109/TSE.2005.85

Function Point Users Group. (2003). *IFPUG Function Point Counting Practices Manual*, Release 4.1.

Garion, C., & Van der Torre, L. (2003, October). Design by contract-Deontic design language for component-based systems. In *Proceedings of the 15th Belgium-Netherlands Conference on Artificial Intelligence (BNAIC2003).* Nijmegen, Netherlands

Gencel, C., & Demirors, O. (2008). Functional size measurement revisited.*ACM Transactions on Software Engineering and Methodology (TOSEM)*, *17*(3), 15.

Hastings, T. E., & Sajeev, A. S. M. (2001). A vector-based approach to software size measurement and effort estimation. *Software Engineering, IEEE Transactions on*, *27*(4), 337-350.

Heričko, M., Rozman, I., & Živkovič, A. (2006). A formal representation of functional size measurement methods. *Journal of Systems and Software*, *79*(9), 1341-1358. doi:10.1016/j.jss.2005.11.568

Jazequel, J. M., & Meyer, B. (1997). Design by contract: The lessons of Ariane. *Computer*, *30*(1), 129-130. doi: 10.1109/2.562936

Kang, K. C., Lee, J., & Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software*, *19*(4), 58-65. doi: 10.1109/MS.2002.1020288

Kiebusch, S., Franczyk, B., & Speck, A. (2005, May). Metrics for software system families. In Proceedings of the *EDSER '05 Proceedings of the seventh international workshop on Economics-driven software engineering research*, (pp. 1-5). Missouri, USA.

Lavazza, L., & Del Bianco, V. (2009, November). A case study in COSMIC functional size measurement: The rice cooker revisited. In *Software Process and Product Measurement* (pp. 101-121). Amsterdam, Netherlands.

Lehne, A. (1997, October). Experience report: function points counting of object oriented analysis and design based on the OOram method. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'97). Atlanta, Georgia.*

Levesque, G., Bevo, V., & Cao, D. T. (2008, May). Estimating software size with UML models. In *Proceedings of the 2008 C3S2E conference* (pp. 81-87). Montreal, Canada.

Lind, K., Heldal, R., Harutyunyan, T., & Heimdahl, T. (2011, November). CompSize: Automated size estimation of embedded software components. In*Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)* (pp. 86-95). Nara, Japan.

Matinlassi, M. (2004, May). Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. In *Proceedings of the 26th International Conference on Software Engineering* (pp. 127–136). Scotland, UK.

McGarry, J. (2001). When it comes to measuring software, every project is unique. *IEEE Software*, *18*(5), 19-21.

Meyer, B. (1997). *Object-oriented software construction.* New York: Prentice-Hall.

Molokken, K., & Jorgensen, M. (2003, September). A review of software surveys on software effort estimation. In *Proceedings of the 2003 International Symposium on Empirical Software Engineering* (pp. 223-230). Rome, Italy.

Nagano, S. I., & Ajisaka, T. (2003, September). Functional metrics using COSMIC-FFP for object-oriented real-time systems. In *13th International Workshop on Software Measurement (IWSM)* (pp. 1-7). Montreal, Canada.

Oligny, S., Abran, A., & Symons, C. (2000, October). COSMIC-FFP some results from the field trials. In *Proceedings* of *15th International Forum on COCOMO and Software Cost Estimation.* Los Angeles, USA.

OMG. (2006). *Unified Modeling Language: Infrastructure, version 2.0.* Retrieved from: http://www.omg.org/spec/UML/2.0/

Ozkan, B., & Demirors, O. (2009, November). Formalization Studies in Functional Size Measurement: How Do They Help?. In *Software Process and Product Measurement* (pp. 197-211). Berlin, Germany.

Poels, G. (2003, April). Functional size measurement of layered conceptual models. In *Proceedings of the 5th International Conference on Enterprise Information Systems, (ICEIS 3),* (pp. 411-416). Angers, France.

Pressman, R. (2005). *Software engineering: a practitioner's approach*. New York: McGrow-Hill International Edition.

Rahman, A. (2004). *Metrics for the structural assessment of product line architecture.* Master's thesis, School of Engineering, Blekinge Institute of Technology.

Sikora, E., Tenbergen, B., & Pohl, K. (2011). Industry needs and research directions in requirements engineering for embedded systems. *Requirements Engineering, 17*(1), 57-78. doi: 10.1007/s00766-011-0144-x.

Soubra, H., Abran, A., Stern, S., & Ramdan-Cherif, A. (2011, November). Design of a Functional Size Measurement Procedure for Real-Time Embedded Software Requirements Expressed using the Simulink Model. In *Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA)* (pp. 76-85). IEEE.

Symons, C. R. (1988). Function point analysis: difficulties and improvements. *IEEE Transactions on Software Engineering, 14*(1), 2-11. doi: 10.1109/32.4618

Symons, C. (2001, May). Come Back Function Point Analysis (Modernized)–All is Forgiven!). In *Proceedings of the 4th European Conference on Software*

Measurement and ICT Control, FESMA-DASMA (pp. 413-426). Heidelberg, GERMANY

Symons, C. R. (2001). Software Benchmarking: Serious Management Tool or a Joke? *IEEE Software*, *18*(5), 18-19.

Vickers, P. (1998). *An Introduction to Function Point Analysis*. Retrieved February 14, 2014, from Northumbria University, School of Informatics Web site: http://computing.unn.ac.uk/staff/cgpv1/downloadables/fpa.pdf

Voelter, M., & Groher, I. (2007, September). Product line implementation using aspect-oriented and model-driven software development. In Proceedings of the *11th International Software Product Line Conference*, (pp. 233-242). Kyoto, Japan.

Vogelezang, F., Symons, C., Lesterhuis, A., Meli, R., & Daneva, M. (2013, October). Approximate COSMIC functional size--guideline for approximate COSMIC functional size measurement. In *Proceedings of* 2013 *Joint Conference of the 23rd International Workshop on Software Measurement* (pp. 27-32). Ankara, Turkey.

Yin, R. K. (2014). *Case study research: Design and methods*. California: Sage publications.

Zubrow, D., & Chastek, G. (2003). *Measures for software product lines* (No. CMU/SEI-2003-TN-031). Carnegie Mellon University. Retrieved from: https://resources.sei.cmu.edu/asset_files/technicalnote/2003_004_001_14195.pdf

APPENDICES

APPENDIX – A SURVEY

1. Purpose of the Survey

What we try to do is to automate the functional size measurement procedure in a component based software product line environment by extracting the required information (functional processes, data movements etc.) from the UML diagrams. The architecture of the product line we are working on is structured by the interface-based design method in which the interfaces of a component are previously defined.

We kindly recommend your valuable opinions about automating the procedure by UML diagrams. Please do not hesitate to ask any details about COSMIC measurement method since the explanation is very short. Thank you for your participation and help.

The comments you have made and your personal information will be kept confidential and will only be used for research purposes.

2. Brief Explanation About COSMIC Software Size Measurement Method

COSMIC FSM was first introduced by The Common Software Measurement International Consortium as a new version of FSM method. The COSMIC measurement method is about applying a set of rules, processes and principles to the Functional User Requirements (FUR) of the software to be measured which outputs a numerical value representing the functional size of the software. The functional size measured by the COSMIC FSM method is independent of implementation decisions whether the software is embedded or not.

In COSMIC, the basic functional components are data movements. The unit of measure is a COSMIC Function Point (CFP) which refers to a movement of the data attributes belonging to a single data group. Data movements can be of four types:

Entry, Exit, Read or Write. The functional process is a set of functional user requirements triggered by an event via an actor – the 'functional user'. The triggering event is an event occurring outside the boundary of the measured software that causes a functional user to initiate a functional process. A functional process comprises at least two data movements: an Entry plus at least either an Exit or a Write.

Figure 35 below illustrates the data movement types and their relationship with the functional process and data groups.

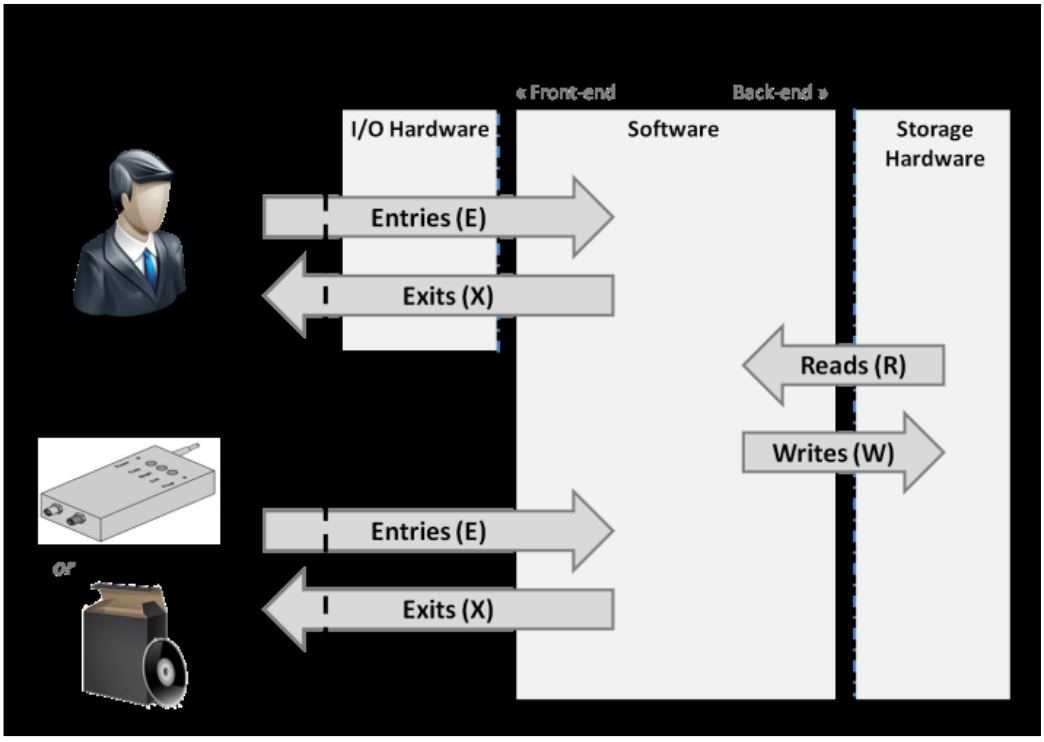

Figure 35 - Data Movement Types

- An Entry (E) moves a data group from a functional user across the boundary into the functional process where it is required.
- An Exit (X) moves the data group from the functional process across the boundary to the functional user where it is required.
- A Read (R) is a data movement that moves a data group from persistent storage to functional process where it is used.
- A Write (W) is a data movement that moves a data group from the functional process to the persistent storage where it is stored.

To calculate the COSMIC Function Points the numbers of the data movements are counted in each functional process. The functional sizes of each data movement type are then added up to have a single functional size.

$$\text{Size (functional process}_i) = \Sigma \text{ size (Entries}_i) + \Sigma \text{ size (Exits}_i) + \Sigma \text{ size (Reads}_i) + \Sigma \text{ size (Writes}_i)$$

3. An Automated Software Size Estimation Approach for Software Product Lines Survey

1. Number of the participant ( )
2. How long have you been a software engineer? ( )
3. In your opinion, how important is software size measurement in the earlier phases of a project? (1 Not Important – 5 Very Important)
   (1) (2) (3) (4) (5)
4. Have you ever measured the software size? ( ) (Y for Yes N for No)
5. Would you measure the software size if it was an automated procedure using the UML diagrams you mainly use for designing the software? ( ) (Y for Yes N for No)
6. The COSMIC functional size measurement and UML concept relation is given in the table below. Can you state which UML diagrams may provide each COSMIC requirement? (Multiple diagram names can be stated in each row)

COSMIC	**UML Concept**	**UML Diagrams**
Software Boundary	Boundary of the component	( ) Sequence Diagram

		( ) Composite structure diagram ( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Functional User	The external components that are directly linked to the ports of the component	( ) Sequence Diagram ( ) Composite structure diagram ( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Triggering Event	Incoming messages to the software boundary	( ) Sequence Diagram ( ) Composite structure diagram

		( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Entry	The functions and events in the provided interface of the component	( ) Sequence Diagram ( ) Composite structure diagram ( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Exit	The functions and events in the required interface of the component	( ) Sequence Diagram ( ) Composite structure diagram

		( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Read	The referred attributes in an action	( ) Sequence Diagram ( ) Composite structure diagram ( ) Class Diagram ( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)
Write	The updated attributes in an action	( ) Sequence Diagram ( ) Composite structure diagram ( ) Class Diagram

		( ) Activity Diagram ( ) Use Case Diagram ( ) Component Diagram ( ) Other (Please Specify)

7. Which UML diagrams would be useful to measure the COSMIC function points? (Multiple selection available, in which ways the diagram can be helpful to automate the measurement)

   ( ) Sequence Diagram

   ( ) Composite structure diagram

   ( ) Class Diagram

( ) Activity Diagram

( ) Use Case Diagram

( ) Component Diagram

( ) Other Type of Diagram (Please Specify)

---

8. Would you be encouraged to draw a UML diagram which you don't usually draw in the design stage for automating the software size measurement? ( ) (Y for Yes N for No)

9. Do you think automating the software size measurement is possible with the UML diagrams you use in the design stage? ( ) (Y for Yes N for No)

Thank you for your participation

APPENDIX – B MANUAL MEASUREMENT RESULTS

Table 19 - Component_18 Manual Measurement Results

Functional Process	Entry (E)	Exit (X)	Read (R)	Write (W)
TVZoomChange	3	2	1	1
TVBrightnessChange	3	2	1	1
TVContrastChange	3	2	1	1
TVFocusChange	3	2	1	1
TRMZoomChange	3	2	1	1
TRMBrightnessChange	3	2	1	1
TRMContrastChange	3	2	1	1
TRMFocusChange	3	2	1	1
TVAutoChange	3	2	1	1
TRMAutoChange	3	2	1	1
TVChangeCamType	2	2	0	0
TRMChangeCamType	2	2	0	0
InitializationTV	2	2	0	0
InitializationTRM	2	2	0	0
CloseTV	1	1	0	0
CloseTRM	1	1	0	0
TRMPolarityChange	3	2	1	1
TRMReady	1	1	0	0
InitializationLRF	2	2	0	0
CloseLRF	1	1	0	0
ActivateLaser	1	1	0	0
DeactivateLaser	1	1	0	0
EchoLogicLRF	1	1	0	0
FireLaser	4	4	0	0
CamComm	2	4	2	2
Total Data Movements	**56**	**47**	**13**	**13**
	Total FP	**129 FP**		

Table 20- Component_19 Manual Measurement Results

Functional Process	Entry (E)	Exit (X)	Read (R)	Write (W)
ArmPosition	1	1	0	0
SafePosition	1	1	0	0
ArmGun	3	2	2	2
SafeGun	3	2	2	2
AutoChange	2	3	1	1
ManualChange	2	3	1	1
BattleMode	1	2	0	0
CockMechUsage	2	2	0	0
FireRate	2	2	0	0
PowerOn	2	2	0	0
PowerOff	2	2	0	0
LastFireSensor	2	2	1	0
StartFire	3	2	1	1
StopFire	2	3	1	0
Initialize	2	2	0	0
Closing	1	1	0	0
GetPower	1	0	1	1
FireCount	2	2	1	0
ChangeGun	3	2	1	1
FireDetectionSensor	2	2	1	0
Total Data Movements	**39**	**38**	**13**	**9**
	Total FP	**99 FP**		

Table 21 - Component_20 Manual Measurement Results

Functional Process	Entry (E)	Exit (X)	Read (R)	Write (W)
Flow	3	0	2	2
GroundTemperature	3	0	2	2
Humidity	3	0	2	2
Pressure	3	0	2	2
Temperature	3	0	2	2
Opening	2	4	0	0
Total Data Movements	**17**	**4**	**10**	**10**
	Total FP	**41 FP**		

Table 22 - Component_21 Manual Measurement Results

Functional Process	Entry (E)	Exit (X)	Read (R)	Write (W)
Attitude	3	0	2	2
Position	3	0	2	2
Target	4	0	3	3
Closing	1	1	0	0
Initialization	2	2	0	0
Total Data Movements	**13**	**3**	**7**	**7**
	Total FP	**30 FP**		

Table 23 - Component_22 Manual Measurement Results

Functional Process	Entry (E)	Exit (X)	Read (R)	Write (W)
ButtonChange	3	1	2	2
CommutatorChange	3	1	2	2
SwitchChange	3	1	2	2

Power	6	4	2	2
Initialization	2	2	0	0
LedsChange	4	1	2	2
HatchChange	3	1	1	1
LimitChange	2	2	0	0
FireAuthorization	1	1	0	0
Total Data Movements	**27**	**17**	**11**	**11**

	Total FP	**63 FP**

APPENDIX – C AUTO MEASUREMENT RESULTS

Table 24 - Automated Measurement Details of Component_18

Functional Process	Triggering Event	Data Movements	E	X	R	W	FP
DayTVBrightnessRequest	evBrightnessReq	(E) evCxfRawDataInd (E) getCameraControlParameters (X) evBrightnessInd (X) getNumberOfBytes (X) peep (X) moveAhead (X) write (E) evBrightnessReq (W)writeBrightnessValue (R)readBrightnessValue	3	5	1	1	10
DayTVContrastRequest	evContrastReq	(X) write (E) evContrastReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evContrastInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeContrastValue (R)readContrastValue	3	5	1	1	10
DayTVFocusRequest	evFocusReq	(X) write (E) evFocusReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evFocusInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeFocusValue (R)readFocusValue	3	5	1	1	10
DayTVZoomRequest	evZoomReq	(X) write (E) evZoomReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evZoomInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeZoomValue (R)readZoomValue	3	5	1	1	10

ThermalBrightnessRequest	evBrightnessReq	(E) evCxfRawDataInd (E) getCameraControlParameters (X) evBrightnessInd (X) getNumberOfBytes (X) peep (X) moveAhead (X) write (E) evBrightnessReq (W)writeBrightnessValue (R)readBrightnessValue	3	5	1	1	10
ThermalContrastRequest	evContrastReq	(X) write (E) evContrastReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evContrastInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeContrastValue (R)readContrastValue	3	5	1	1	10
ThermalFocusRequest	evFocusReq	(X) write (E) evFocusReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evFocusInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeFocusValue (R)readFocusValue	3	5	1	1	10
ThermalZoomRequest	evZoomReq	(X) write (E) evZoomReq (E) evCxfRawDataInd (E) getCameraControlParameters (X) evZoomInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeZoomValue (R)readZoomValue	3	5	1	1	10
DayTVAutoRequest	evSetCameraModeAutoReq	(X) write (E) evSetCameraModeAutoReq (E) evCxfRawDataInd (X) evSetCameraModeAutoInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeAutoMode (E) getCameraControlParameters (R)readAutoMode	3	5	1	1	10

ThermalAutoRequest	evSetCameraModeAutoReq	(X) write (E) evSetCameraModeAutoReq (E) evCxfRawDataInd (X) evSetCameraModeAutoInd (X) getNumberOfBytes (X) peep (X) moveAhead (W)writeAutoMode (E) getCameraControlParameters (R)readAutoMode	3	5	1	1	10
DayTVChangeCamTypeRequest	evAS3ImageSelectionReq	(E) evCxfRawDataInd (X) getNumberOfBytes (X) peep (X) moveAhead l (X) write (E) evAS3ImageSelectionReq (X) evAS3ImageSelectionInd	2	5	0	0	7
ThermalChangeCamTypeRequest	evAS3ImageSelectionReq	(E) evCxfRawDataInd (X) getNumberOfBytes (X) peep (X) moveAhead l (X) write (E) evAS3ImageSelectionReq (X) evAS3ImageSelectionInd	2	5	0	0	7
DayTVClosingScenario	evOffReq	(E) evOffReq (X) evOffInd	1	1	0	0	2
ThermalClosingScenario	evOffReq	(E) evOffReq (X) evOffInd	1	1	0	0	2
DayTVInitializationCamera	evStartReq	(X) evStartInd (E) evOnReq (X) evOnInd (E) evStartReq	2	2	0	0	4
ThermalInitializationCamera	evStartReq	(X) evStartInd (E) evOnReq (X) evOnInd (E) evStartReq	2	2	0	0	4
AS3CommunicationScenario	evBITResultInd	(X) evErrorInd (X) evErrorInd (X) evBITResultInd (E) getBITErrors (E) getBITErrors (X) evBITResultInd (W)writeInitializationStatus (R)readInitializationStatus (W)writeCommStatus (R)readCommStatus	2	4	2	2	10

LRFActivationRequest	evLRFActivateReq	(X) evLRFActivateInd (E) evLRFActivateReq	1	1	0	0	2
LRFClosingScenario	evOffReq	(E) evOffReq (X) evOffInd	1	1	0	0	2
LRFDeactivationRequest	evLRFInActivateReq	(X) evLRFInActivateInd (E) evLRFInActivateReq	1	1	0	0	2
LRFEchoLogicRequest	evSetEchoLogicReq	(X) evSetEchoLogicInd (E) evSetEchoLogicReq	1	1	0	0	2
LRFFireRequest	evLRFFireReq	(X) write (E) evCxfRawDataInd (X) evLRFFiredInd (X) getNumberOfBytes (X) peep (X) moveAhead (E) evLRFFireReq (E) evLRFFireStopReq (X) write (E) evCxfRawDataInd (X) evLRFFireStopInd	4	7	0	0	11
LRFInitialization	evStartReq	(X) evStartInd (E) evStartReq (X) evOnInd (E) evOnReq	2	2	0	0	4
ThermalPolarityRequest	evCameraCommandReq	(X) evCameraCommandInd (E) evCxfRawDataInd (X) getNumberOfBytes (X) peep (X) moveAhead (E) evCameraCommandReq (X) write (E) getCameraControlParameters (W)writePolarityType (R)readPolarityType	3	5	1	1	10
ThermalReady	evCxfRawDataInd	(X) evAS3IRReadyInd (E) evCxfRawDataInd (X) getNumberOfBytes (X) peep (X) moveAhead	1	4	0	0	5
			56	**92**	**13**	**13**	**174**

Table 25 - Automated Measurement Details of Component_19

Functional Process	Triggering Event	Data Movements	E	X	R	W	FP
GSABInitialization	evStartReq	(E) evStartReq (X) evStartInd (E) evOnReq (X) evOnInd	2	2	0	0	4
PowerOnRequest	evPowerOnReq	(X) evPowerOnInd (E) evPowerOnReq (E) evCxfCanDataInd (X) write	2	2	0	0	4
PowerOffRequest	evPowerOffReq	(X) evPowerOffInd (E) evPowerOffReq (E) evCxfCanDataInd (X) write	2	2	0	0	4
BattleModeRequest	evSetBattleModeReq	(X) write (E) evSetBattleModeReq (X) evSetBattleModeInd	1	2	0	0	3
ArmedPositionRequest	evSetCockingMechanismArmedPositionReq	(X) write (E) evSetCockingMechanismArmedPositionReq	1	1	0	0	2
SafePositionRequest	evSetCockingMechanismSafePositionReq	(E) evSetCockingMechanismSafePositionReq (X) write	1	1	0	0	2
CockingMechanismUsageRequest	evSetCockingMechanismUsageReq	(E) evSetCockingMechanismUsageReq (X) write (X) evSetCockingMechanismUsageInd (E) evCxfCanDataInd	2	2	0	0	4

FireCountRequest	evSetFireRateReq	(X) write (X) evSetFireRateInd (E) evSetFireRateReq (R)readFiringStatus (E) evCxfCanDataInd	2	2	1	0	5
FireRateRequest	evSetFireCountReq	(X) write (E) evSetFireCountReq (R)readFiringStatus (E) evCxfCanDataInd (X) evSetFireCountInd	2	2	1	0	5
FireDetectionSensorUsageRequest	evSetFireDetectionSensorUsageReq	(X) write (E) evSetFireDetectionSensorUsageReq (X) evSetFireDetectionSensorUsageInd (R)readGunManualStatus (E) evCxfCanDataInd	2	2	1	0	5
ArmGunRequest	evReadyForFireReq	(R)readCockingMechanismSensorCancelled (E) evCxfCanDataInd (X) evReadyForFireInd (X) write (E) evReadyForFireReq (R)readManualOrAutoMode (W)writeGunArmed (E) isArmed (R)readGunArmed	3	2	3	1	9
SafeGunRequest	evSafeReq	(R)readCockingMechanismSensorCancelled (E) evCxfCanDataInd (X) evSafeInd (E) evSafeReq (X) write (R)readManualOrAutoMode (E) isSafe (W)writeGunSafe (R)readGunSafe	3	2	3	1	9
StartFireRequest	evStartFireCmd	(E) evCxfCanDataInd (W)setPIO (X) evStartFireInd (X) write (E) evStartFireCmd (R)readManualOrAutoMode (E) isFiring (W)writeFiring (R)readFiring	3	2	2	2	9

StopFireRequest	evOffReq	(E) evCxfCanDataInd (W)resetPIO (X) evStopFireInd (X) evFiringCompletedInd (E) evStopFireCmd (X) write (R) readManualOrAutoMode	2	3	1	1	7
AutoIndication	evStartReq	(X) evAutoModeInd (X) evWarningInd (W)writeAutoMode (E) isInManualMode (E) evCxfCanDataInd (R)readAutoOrManual (X) write (R)readAutoMode	2	3	2	1	8
ManualIndication	evCxfCanDataInd	(X) evManualModeInd (X) evWarningInd (W)writeManualMode (E) isInManualMode (R)readAutoOrManual (E) evCxfCanDataInd (X) write (R)readManualMode	2	3	2	1	8
LastFireDetectionSensorUsageRequest	evSetLastFireDetectionSensorUsageReq	(X) evSetLastFireDetectionSensorUsageInd (R)readAutoOrManualMode (E) evSetLastFireDetectionSensorUsageReq (X) write (E) evCxfCanDataInd	2	2	1	0	5
GunTypeChangeRequest	evCxfCanDataInd	(X) evSetGunInd (W)writeGunType (X) write (E) evCxfCanDataInd (E) getGunType (R)readGunType	2	2	1	1	6
GetPowerRequest	getPower	(E) getPower (R)readPower (W)writePower	1	0	1	1	3
GSABClosingScenario	evOffReq	(E) evOffReq (X) evOffInd	1	1	0	0	2
			38	**38**	**19**	**9**	**104**

Table 26 - Automated Measurement Details of Component_20

Functional Process	Triggering Event	Data Movements	E	X	R	W	FP
FlowRequest	getFlow	(E) evCxfRawDataInd (W)saveFlowServiceStatus (E) getFlowServiceStatus (E) getFlow (X) moveAhead (X) peep (X) getNumberOfBytes (W)saveFlow (R)readFlow (R)readFlowServiceStatus	3	3	2	2	10
GroundTemperatureRequest	getTemperature	(E) getTemperatureServiceStatus (X) getNumberOfBytes (W)saveTemperature (E) getTemperature (X) moveAhead (X) peep (W)saveTemperatureServiceStatus (E) evCxfRawDataInd (R)reademperature (R)readTemperatureServiceStatus	3	3	2	2	10
HumidityRequest	getHumidity	(E) evCxfRawDataInd (W)saveHumidityServiceStatus (E) getHumidityServiceStatus (E) getHumidity (X) moveAhead (X) peep (X) getNumberOfBytes (W)saveHumidity (R)readHumdity (R)readHumidityServiceStatus	3	3	2	2	10
PressureRequest	getPressure	(E) evCxfRawDataInd (W)savePressureServiceStatus (E) getPressureServiceStatus (E) getPressure (X) moveAhead (X) peep (X) getNumberOfBytes (W)savePressure (R)readPressure (R)readPressureServiceStatus	3	3	2	2	10

TemperatureRequest	getTemperature	(E) evCxfRawDataInd (W)saveTemperatureServiceStatus (E) getTemperatureServiceStatus (E) getTemperature (X) moveAhead (X) peep (X) getNumberOfBytes (W)saveTemperature (R)readTemperature (R)readTemperatureServiceStatus	3	3	2	2	10
IRDAMInitialization	evStartReq	(X) evStartInd (E) evOnReq (X) evOnInd (E) evStartReq (X) configure (X) open	2	4	0	0	6
			17	**19**	**10**	**10**	**56**

Table 27 - Automated Measurement Details of Component_21

Functional Process	**Triggering Event**	**Data Movements**	**E**	**X**	**R**	**W**	**FP**
InitializationBoomerang	evStartReq	(X) evOnInd (E) evOnReq (X) evStartInd (E) evStartReq (E) evOffReq (X) evOffInd	3	3	0	0	6
GetAttitude	evCxfRawDataInd	(W)writeAttitudeServiceStatus (X) peep (E) evCxfRawDataInd (E) getAttitute (E) getAttituteServiceStatus (W)writeAttitude (X) moveAhead (R)readAttitude (R)readAttitudeServiceStatus	3	2	2	2	9

GetPosition	evCxfRawDataInd	(W)writePositionInfoServiceStatus (X) peep (E) evCxfRawDataInd (E) getPositionInfo (E) getPositionInfoServiceStatus (W)writePositionInfo (X) moveAhead (R)readPositionInfo (R)readPositionServiceStatus	3	2	2	2	9
GetTarget	evCxfRawDataInd	(W)writeTargetInfoServiceStatus (E) getTargetList (W)writeTargetList (X) peep (E) evCxfRawDataInd (E) getTargetInfo (E) getTargetServiceStatus (W)writeTargetInfo (X) moveAhead (R)readTargetList (R)readTargetServiceStatus (R)readTargetInfo	4	2	3	3	12
			13	**9**	**7**	**7**	**36**

Table 28 - Automated Measurement Details of Component_22

Functional Process	**Triggering Event**	**Data Movements**	**E**	**X**	**R**	**W**	**FP**
InitializationSKB	evStartReq	(X) evStartInd (E) evStartReq (X) attach (X) attach (X) attach (E) evOnReq (X) evOnInd	2	5	0	0	7
PowerOffRequest	evPowerOffReq	(E) evPowerOffReq (E) evCxfCanDataInd (X) evPowerOffInd (E) getPower (X) write (W)writePowerStatus (R)readPowerStatus	3	2	1	1	7

PowerOnRequest	evPowerOnReq	(E) evPowerOnReq (E) evCxfCanDataInd (X) evPowerOnInd (E) getPower (X) write (W)writePowerStatus (R)readPowerStatus	3	2	1	1	7
SwitchStatusChange	evCxfCanDataInd	(X) evSwitchStatusInd (E) getSwitchStatus (E) getSwitchServiceStatus (E) evCxfCanDataInd (W)writeSwitchStatus (R)readSwitchStatus (R)readSwitchServiceStatus	3	1	2	1	7
SetLedRequest	evSetLedReq	(E) evSetLedReq (E) getLedStatus (E) getLedServiceStatus (X) write (W)writeLedStatus (W)writeLedServiceStatus (R)readLedStatus (R)readLedServiceStatus (E) evCxfCanDataInd	4	1	2	2	9
ButtonStatusChange	evCxfCanDataInd	(E) getButtonStatus (E) getButtonServiceStatus (E) evCxfCanDataInd (W)writeButtonStatus (R)readButtonStatus (R)readButtonServiceStatus (W)writeButtonServiceStatus	3	0	2	2	7
CommuatorStatusChange	evCxfCanDataInd	(E) evCxfCanDataInd (X) evErrorInd (E) getCommutatorStatus (E) getCommutatorServiceStatus (X) evCommutatorStatusInd (W)writeCommutatorStatus (R)readCommutatorStatus (R)readCommutatorServiceStatus (W)writeCommutatorStatus (W)writeCommtatorServiceStatus	3	2	2	3	10

ElectricalLimitsOverrideRequest	evElectricalLimitsOverrideReq	(E) evElectricalLimitsOverrideReq (X) evElectricalLimitsOverrideInd (X) write (E) evCxfCanDataInd	2	2	0	0	4
FireAuthorizationSelectionRequest	evFireAuthorizationSelection	(E) evFireAuthorizationSelection (X) write	1	1	0	0	2
HatchWarningOverrideRequest	evHatchWarningOverride	(X) write (E) getHatchStatus (E) evHatchWarningOverride (W)writeHatchStatus (R)readHatcStatus (E) evCxfCanDataInd	3	1	1	1	6
			27	**17**	**11**	**11**	**66**

ENFORMATİK ENSTİTÜSÜ
GRADUATE SCHOOL OF INFORMATICS

ORTA DOĞU TEKNİK ÜNİVERSİTESİ
MIDDLE EAST TECHNICAL UNIVERSITY

DUMLUPINAR BULVARI 06800
ÇANKAYA ANKARA/TURKEY
T: +90 312 210 37 41
F: +90 312 210 37 45
contact@ii.metu.edu.tr
www.ii.metu.edu.tr

21.08.2014

Sayı : 59473358 / 726-4739

GÖNDERİLEN: Prof.Dr.Belgin Ayvaşık
Rektör Danışmanı

GÖNDEREN: Dr.Ali Arifoğlu
Enformatik Enstitüsü Müdür V.

KONU: Önder Eren

Bilişim Sistemleri Anabilim Dalı Yüksek Lisans programı öğrencisi 1696517 no.lu Önder Eren'in, 15 Ağustos 2014 – 30 Ağustos 2014 tarihleri arasında "An Automated Software Size Estimation Approach for Software Product Lines" başlıklı yüksek lisans tezi çalışmasına ilişkin "Aselsan ve SST Direktörlüğü"nde uygulama yapmak için görevlendirme başvurusu incelenmiş, ilgili EABD Başkanlığı'nın görüşüne dayanarak adı geçen öğrencinin isteği doğrultusunda görevlendirilmesine Etik Komite onayı koşulu ile uygun görülmüştür.

Saygılarımla,

Ek: YKK
EABD

Etik Komite Onayı

Uygundur

25../08/2014

Bilgi ve gereği ricasıyla

26/08

Prof. Dr. Belgin AYVAŞIK
Rektör Danışmanı

Canan ÖZGEN
Uygulamalı Etik Araştırma Merkezi
(UEAM) Başkanı
ODTÜ 06531 ANKARA

541.
27/8.

26/8. Yazının aslını Önder Eren adına elden aldım.
Nihan OCAK
29.08.2014

INFORMATICS INSTITUTE

TEZ FOTOKOPİSİ İZİN FORMU

ENSTİTÜ

Fen Bilimleri Enstitüsü ☐

Sosyal Bilimler Enstitüsü ☐

Uygulamalı Matematik Enstitüsü ☐

Enformatik Enstitüsü ☐

Deniz Bilimleri Enstitüsü ☐

YAZARIN

Soyadı : ……
Adı : ……
Bölümü : ……...

TEZİN ADI (İngilizce) : ……....
……
……
……
……...

TEZİN TÜRÜ : Yüksek Lisans ☐ Doktora ☐

1. Tezimin tamamından kaynak gösterilmek şartıyla fotokopi alınabilir. ☐
2. Tezimin içindekiler sayfası, özet, indeks sayfalarından ve/veya bir bölümünden kaynak gösterilmek şartıyla fotokopi alınabilir. ☐
3. Tezimden bir (1) yıl süreyle fotokopi alınamaz. ☐

TEZİN KÜTÜPHANEYE TESLİM TARİHİ : ……