A FUNCTIONAL SOFTWARE MEASUREMENT APPROACH BRIDGING THE GAP BETWEEN PROBLEM AND SOLUTION DOMAINS

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

BY

ERDİR UNGAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

NOVEMBER 2013

A FUNCTIONAL SOFTWARE MEASUREMENT APPROACH BRIDGING THE GAP BETWEEN PROBLEM AND SOLUTION DOMAINS

Submitted by **ERDİR UNGAN** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in 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**

Examining Committee Members:

Prof. Dr. Semih Bilgen _____
Electrical and Electronics Engineering Dept., METU

Prof. Dr. Onur Demirörs _____
Information Systems, METU

Assist. Prof. Dr. Aysu Betin Can _____
Information Systems, METU

Assoc. Prof. Dr. Altan Koçyiğit _____
Information Systems, METU

Assist. Prof. Dr. Özgür Tanrıöver _____
Computer Engineering Dept., Ankara University

Date: 8.11.2013

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 work.

Name, Last Name : Erdir Ungan

Signature : _____

ABSTRACT

A FUNCTIONAL SOFTWARE MEASUREMENT APPROACH BRIDGING THE GAP BETWEEN PROBLEM AND SOLUTION DOMAINS

Ungan, Erdir

Ph. D., Department of Information Systems

Supervisor: Prof. Dr. Onur Demirörs

November 2013, 121 pages

There are various software size measurement methods that are used in various stages of a software project lifecycle. Although functional size measurement methods and lines of code measurements are widely practiced, none of these methods explicitly position themselves in problem or solution domain. This results in unreliable measurement results as abstraction levels of the measured artifacts vary greatly. Unreliable measurement results hinder usage of size data in effort estimation and benchmarking studies. Furthermore, there exists no widely accepted measurement method for solution domain concepts other than lines of code, such as software design. In this study, an approach is defined to distinguish problem and solution domains for a software project and a software size measurement methodology for solution domain is proposed based on software design sizes.

Keywords: Functional Size Measurement, Software Design Size, Estimation, Decomposition

öz

PROBLEM VE ÇÖZÜM UZAYI ARASINDA BAĞLANTI SAĞLAYAN BİR İŞLEVSEL BÜYÜKLÜK ÖLÇÜM YAKLAŞIMI

Ungan, Erdir

Doktora, Bilişim Sistemleri

Tez Yöneticisi: Prof. Dr. Onur Demirörs

Kasım 2013, 121 sayfa

Yazılım proje yaşam döngüsünün çeşitli aşamalarında kullanılan birçok yazılım büyüklük ölçüm metot bulunmaktadır. İşlevsel büyüklük ölçümü ile kod satır sayısı tabanlı ölçümler yaygın olarak kullanılsa da, bu yöntemlerin hiçbiri kendisini kesin ve açık bir şekilde problem veya çözüm uzayında konumlamamaktadır. Ölçülen kavramların soyutluk seviyeleri büyük değişiklikler gösterdiğinden, bu durum, ölçüm sonuçlarının güvenilirliğini azaltmaktadır. Güvenilir olmayan ölçüm sonuçları ise, ölçüm verilerinin, işgücü kestirimi, kıyas çalışmaları gibi alanlarda kullanılmasını zorlaştırmaktadır. Bunun yanında, yazılım tasarımı gibi kod satır sayısı dışındaki çözüm uzayı kavramları için yaygın olarak kabul görmüş ölçüm yöntemleri bulunmamaktadır. Bu çalışmada, problem ve çözüm uzaylarının ayrıştırılması için bir yaklaşım önerilmiş ve çözüm uzayı için, yazılım tasarım büyüklüklerini temel alan bir yazılım büyüklük ölçüm yöntemi önerilmiştir.

Anahtar Kelimeler: İşlevsel Büyüklük Ölçümü, Yazılım Tasarım Büyüklüğü, Kestirim, İşlevsel Kırılım

To my mother, to whom I owe everything good I have in life

ACKNOWLEDGEMENTS

I first, want to express my thanks to my Ph.D. supervisor, Onur Demirörs. His positive attitude, trust and support made it possible for me to go through the process. I would have been lost without his ideas and encouragement.

My sincere thanks to Özden Hanoğlu, Özden Özcan Top, Banu Aysolmaz, Barış Özkan, Gökçen Yılmaz and many other friends at Software Management Research Group. This research would not have been possible without their collaboration and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Anything you try to quantify can be divided into any number of "anythings," or become the thing - the unit - itself. And what is any number, itself, but just another unit of measurement? What is a 'six' but two 'threes', or three 'twos'...half a 'twelve', or just six 'ones' - which are what?

- F.L. Vanderson

Software projects are conducted to solve a problem in the real life. Similar to the case in other engineering disciplines, it is possible to develop multiple solutions to a project. As these solutions may differ greatly, their size and the effort required to realize them also vary significantly.

This fact, makes it difficult to establish a direct relationship between the product and the process to develop that product. As there is no one to one relationship between a problem and its possible solutions, it is difficult to define a relation between the size of the problem and the solution.

Within the discipline of software measurement, there exists methods that measure both problem and solution domains. Problem domain measurements came a long way since their initiation and can quantify problem definitions and specifications. Solution domain measurements are more formal and more precise as they are based on physical constructs and models.

However, problem domain measures fall short in accuracy as inputs to prediction models, and solution domain measures emerge too late in a project lifecycle to be used in predictions. I believe this is one of the main problems in software measurement (and estimation) as a discipline.

The main reason for problem domain measurement's failure in representing the development effort lies in the ambiguity of the process of developing a solution to a problem at hand. Developing a solution to a problem is a "soft" area. Therefore problem domain concepts fail to predict solution domain concepts on their own.

Most of the measurement and estimation methods assume that there is a continuity in the development lifecycle, which begins with the problem statement and ends with development and testing of the software product.

However, there is an inherent discontinuity between the concepts of problem and solution. There is a gap between these two domains which the actual engineering or "art" as some would call traverses.

Jackson states that, the solution is an answer from the machine domain to the problem [1]. Therefore the relation between these two domains is not straightforward. It is affected by the designer's skills, imagination, and experience. Certain factors such as use of design patterns, similarities between problem-solution tuples in an organization's historical data and traditions of an organization tend to help making this problem to solution transformation formal and algebraic. Most of the estimation methods in the literature exploit these factors and try to calculate solution domain concepts such as development effort using certain multipliers for external factors and/or curve fitting algorithms.

Measurements in problem domain are good for representing problem domain concepts and measurements in solution domain are good for representing solution domain concepts. Problem domain concepts such as price, features, can be represented by problem domain measurements such as function points, feature points, use case points. Solution domain concepts such as development effort, physical size, and developer performance can be represented in solution domain sizes such as LOC and design based sizes. See section 3.2

The capability to accurately quantify the size of software at an early stage of the development lifecycle is critical to software project managers for evaluating risks, developing project estimates (e.g., effort, cost) and having early project indicators (e.g., productivity).

In the changing software industry, the rate of releases is higher, and the extent of what is deployed in each release is smaller.

The development and the maintenance of software systems accounts for one percent of the world's economy [87]. However, according to a recent Standish group research study [92], only an average of 34% of software projects are completed on time and within budget. Therefore, 32% of projects failed due to over-time, over-budget or

cancelations. In addition, this study pointed out that on average only 61% of the originally specified functionality was delivered to customers.

In this context, software size measurement plays an important role. It is widely accepted that software size is one of the key factors that potentially affect the cost and time of the software projects [44] [93][38] [49][43].

Usefulness of functional size measurement

Software size measurement is an important part of the software development process Functional size measures are used to assess the logical external view of the software from the users' perspective by measuring the amount of functionality to be delivered. These measures can be used for a variety of purposes, such as project estimation [62][43], quality assessment [54], benchmarking [56], outsourcing contracts [77].

According to ISO/IEC 14143-1 [69] functional size measurements can be used for:

- Budgeting software development or maintenance
- Tracking the progress of a project
- Negotiating modifications to the scope of the software
- Determining the proportion of the functional requirements satisfied
- by a software package
- Estimating the total software asset of an organization
- Managing the productivity of software development, operation or
- maintenance processes
- Analyzing and monitoring software defect density.

The use of functional size measures has been extensively discussed in the literature.

Fetcke et al. [96]proposed a generalized representation for functional size measurement that captures the main concepts used by FSM methods to represent a system so that its functional size is emphasized. This representation was applied to IFPUG FPA [66], Mark II FPA [100], NESMA and COSMIC [2].

All these methods can be characterized as data-oriented. According to this model, functional size measurement requires two steps of abstraction, called the identification step and the measurement step.

The aim of the identification step is to identify the elements in the requirements documentation that add to the functional size of the system. The result of this first abstraction activity is an abstract model of the relevant elements for functional size measurement, according to the metamodel of the FSM method that is used.

- User concept: the users that interact with a system. These can be human users or software and hardware.
- Application concept: the object of measurement. Applications provide functions to the users.
- Transaction concept: the processes of interaction between the user and the system from a "logical" perspective.
- Data concept: the data stored by the system. Data elements represent the smallest data items that are meaningful to the user. Data elements can be structured in logically related groups of data.

During the measurement step, the elements in the abstract model are mapped into numbers representing the (relative) amount of functionality that is contributed to the functional size of the system. Finally, the numbers are aggregated into an overall functional size value.

1.1. Problem Statement

In software project management it is crucial to be able to accurately quantify the size of software as the size information is utilized as an input in most of the management activities such as developing project estimates (e.g., effort, cost), risk assessment, productivity measurement, performance management, benchmarking, quality management.

We base our measurement approach on functional size measurement (FSM) methods. However, so far several limitations and problems about functional size measurements have been reported. Below, we list problems which we believe of highest importance.

1.1.1. Problems of Reliability

One other serious problem for functional measurement methods is accuracy of measurement results in reflecting the true size of a software product and repeatability issues among measurers. There exist several studies assessing the reliability issues of FSM methods [3][49][51][43]. We have also conducted a series of experiments in order to point out accuracy and repeatability problems in COSMIC. We found that measurement results for a single requirements set can vary greatly among different measurers. Most of the measurers, especially inexperienced measurers, also fail to get a measurement result inside an acceptable error range.

We also investigated root causes of discrepancies and measurement errors and found that:

- Errors are common in:
    - Identification of Functional Processes

    - Identification of Objects of Interest
    - Identification of Data Groups
- Ambiguous requirements result in big differences in measurement results.
- Assumed solution choices and/or individual's interpretations cause discrepancies in measurement results.
- Imperfect or incomplete definitions of a system result in wrong overall system size instead of measuring only the defined parts of a system.

1.1.2. Problems of Granularity

Most, if not all, measurement methods in problem domain does not incorporate a definition for abstraction and/or granularity level of the system being measured. FSM methods such as COSMIC do define the granularity level of functionality to be measured but they lack a definition of granularity for the system itself.

Today, software systems can be so large that it became virtually impossible to define and communicate a whole system within a single analysis. This applies to both functional and structural aspects of the definition of a system. Big software systems are now defined in various levels of decomposition. Correspondingly, their functionality is defined in various levels. Their architecture is vertically decomposed in many layers as well as horizontally into components.

Development of subsystems resulting from the decomposition of a bigger system can be delegated to separate development teams, departments or even companies. Software projects are defined for intermediate software products such as services, layers and components. These projects can be defined so disjointed that there may be no information available to identify the decomposition level of the software product. The end product can be a subsystem of a bigger system which is in turn a subsystem of a bigger system. Similarly, the software product can be utilizing smaller systems which in turn utilize further smaller systems themselves.

As problem domain software measures, that is predominantly FSM methods, lack the information about the granularity level of the system being measured, size of a super system, sub system or a component are all represented in a single level. Making, measurements non additive, non homomorphic and non transitive. This violates the metrological requirements of a measurement. With existing measurement methods, size of an overall system will be different from their total size and cannot be calculated from the size of its subordinates.

Apart from metrological problems, lack of levels in measurement result introduces arbitrariness to their relation with real life concepts and their measured values such as development effort, crippling their adequacy to be utilized in estimation of those values.

Moreover, based on decomposition, certain software projects rely on much lower level specifications than functional user requirements to define the required functionality of their software product. However, problem domain software measures, that is predominantly FSM methods, utilize functional user requirements as the main input for sizing functionality.

FSM methods such as COSMIC address the level of granularity by the definition for Functional Process. However, this definition does not specify a granularity level for the behavior of the system. Instead, it defines how to select a consistent set of actions defined in the Requirement Specifications and group them to be measured separately.

This approach however, assumes that (or should be assuming that) the requirements to be measured are defined at the same level of granularity. It does not define a formal method to use the same level of granularity in the requirements.

The level of granularity and hence, the number of Objects of Interest directly affect the measurement result.

Functional process is not a universal or absolute level of abstraction/granularity. It is a relative definition for granularity as system functionality definitions can be broken down into sub system functionality definitions and one may start to define functional processes with a functionality definition which is either higher or lower in level of abstraction in the continuum of problem-solution domain chain. (See section 2.2)

Defining functional process in each level of abstraction is possible. Rules for defining functional processes can be applied to any level of system definition and therefore the granularity level for the functionality definition will be consistent within the level in which the measurement is performed. However, there will be functional processes defined in different system decomposition levels, making the granularity level defined by function process neither universal nor absolute.

Moreover, as far as software maintenance is concerned, changes request can be in any level of decomposition. That is, changes in a requirement can be measured by FSM methods, however a change request only causing a change in the implementation is hard to measure by using artifacts in higher levels. Similarly, two low level changes would be represented by the same measurement in higher levels. Based on this resolution problem, FSM methods may prove to be inaccurate while to measure lesser than functional user requirement level changes.

1.1.3. Problems in Effort Estimation

During the last thirty years, numerous estimation models for software projects were developed. Besides these generic models, organizations also developed their own, specific estimation models. These models can be classified at the top level as [74];

- Expert Judgment: These methods rely on the opinions of experts who experience in software development within the application domain.
- Formal Estimation Methods: These methods rely on quantitative data and mathematical models and formulas that utilize them.
- Composite Estimation Methods: These methods combine elements of the two methods and include both judgmental and mathematical methods.

However, effort estimation models still far from the required accuracy.

Boehm states [47]:

> *"Today, a software cost estimation model is doing well if it can estimate software development costs within 20% of the actual costs, 70% of the time, and on its own home turf (that is, within the class of projects to which it is calibrated…. This means that the model's estimates will often be much worse when it is used outside its domain of calibration."*

Similarly, Ferens and Christensen state that [60]:

> *"…in general, model validation showed that the accuracy of the models was no better than within 25 percent of actual development cost or schedule, about one half of the time, even after calibration."*

Parametric effort estimation methods in the literature take three main aspects into consideration. Software Size, external factors and subjective assessments.

Some estimation methods such as: COCOMO [4] [5] utilize solution domain measures such as lines of code (LOC) and measurements based on design constructs. Development effort is a concept that lies in the solution domain. Therefore effort estimation methods that get their size input from solution domain measurements are relatively more accurate than those that get the size input from problem domain measurement.

However, in order for an estimation to be useful, it should produce results as early as possible in the lifecycle and solution domain size measurements are not ready until it is too late. Therefore, most of the methods in the literature utilize functional size measurement methods such as Function Points, IFPUG, COSMIC, NESMA and similar

methods [6][4] .These methods measure software size based on requirements and specifications, which is ready at the beginning of the project lifecycle.

Having the functional size at hand, estimation methods define certain external factors that affect development effort. Although different estimation models feature different factors, most common factors can be summarized as [68][78]:

- **Project context:** country, type of organization, business area, and type of development.
- **Product characteristics:** application type, architecture, and user base.
- **Development characteristics:** team characteristics, implementation technology (development language, development platform, tools used), and development techniques.
- **Qualitative factors influencing project execution:** developer experience, project requirements stability, environment, and development tool suitability.

Several studies show that size information obtained by problem domain measurement methods, prove to be inefficient in estimating project effort. In a previous study, we researched the differences between in house and public data sets based on their ability to estimate further projects in an organization. We showed that estimation methods devised based on an organizations historical data have marginal success in effort estimation and those that base on public benchmarking data sets such as ISBSG produce estimation results far beyond acceptable error margins.

On the other hand measurement methods in solution domain, based on source code, design constructs and algorithms generally correlate better with development effort data than those in problem domain [103][7]. However, solution domain sizes cannot be obtained early in the project lifecycle which is the time estimations are actually needed.

In order to overcome this, estimation methods suggest either predicting software size and use those approximate values for estimation or measure domain size and convert it to solution domain size based on some historical data [47][60]. Both these approaches again, introduce errors in estimation. Lastly, estimation methods require some subjective assessment from the user to complete the estimation such as; degree of complexity of software, team experience, suitability of development environment and familiarity. This is actually how these methods try to bridge the gap between the problem domain and solution domain. As I discussed above, the process of developing a solution to a problem is ambiguous and soft. Therefore, estimation methods based on problem domain sizes need some subjective input from the user in order to incorporate the expertise and foresight of the person conducting the estimation. However, this is one of the points where the reliability and repeatability of these methods are questioned. [7]

Most of the software organizations which use parametric effort estimation models use SLOC/FP ratios to predict the code size and then use the value as an input to cost estimation models. However, the results of this study showed that even obtained at the component level, using these ratios can cause significant amount of error. Therefore, we conclude that software organizations should not continue using these ratios unless their local studies show acceptable results.

1.1.4. Problems in Benchmarking

The above mentioned problems in measurement results and granularity also affects the quality of data in benchmarking data sets. Özcan Top and Yilmaz [86] conducted a study in our research group on benchmarking data sets such as ISBSG, and concluded that, those sets lack structural information about the projects. We cannot deduct information about the abstraction level of measurements in those data sets. Due to factors discussed in 1.1.2 comparing data from varying abstraction levels will result in erroneous benchmarking. With existing measurement methods, size of an overall system will be different from their total size and cannot be calculated from the size of its subordinates. And one cannot get the information about how total size for a project is calculated.

Similarly, studies also shown that the quality of the measurement in public data sets is questionable and reliability of any size information is disputed, mostly due to factors defined in section 1.1.1.

1.2. Solution Approach

In this thesis, we recognize the separation of problem and solution domains. We relate activities, artifacts and their corresponding size measurements to these domains.

While separating the problem and solution domains, we also suggest that problem and solution domain definitions shift as problems are decomposed into smaller problems. System decomposition and abstraction techniques guide this shift through the engineering process.

ISO/IEC 14143-1:2007 [69] defines Base Functional Component (BFC) as "An elementary unit of Functional User Requirements defined by and used by an FSM Method for measurement purposes".

The standard specifies that calculation of an FSM must be based on the evaluation of each BFC. ISO/IEC 14143-1:2007 requires, that FSM methods should:

Define the rules to determine the BFCs.

- Define how to assign a numeric value to a BFC according to its BFC Type.

- Calculate the functional size by measuring BFCs.

In functional size measurement, the functional user requirements allocated to one or more pieces of software are represented by functional processes. Each functional user requirement is broken down into one or more functional processes. Further braking down the functionality, functional processes are broken down into sub-processes. A sub-process can either be a data movement or data manipulation.

Data movements and manipulations are the basic functions a software can perform on data or a data group. Data movements occur when a single consistent group of data is moved through a boundary of an encapsulation of any level. It can be a system boundary, component boundary or the encapsulation of an object based on the level of decomposition the functional process is defined.

In functional size measurement, data manipulations are assumed to be inherent within data movements and therefore disregarded for the purposes of measuring the functional size.

Similar to the most common FSM methods, we consider data movements as the base functional component for measurement. Data movement is an abstract, domain independent concept. It is atomic as far as a decomposition level is concerned and can be well defined for a given measurement view.

Abran [8] states this as: "The key concept of functionality at the highest level of commonality that is present in all software was identified as the data movement. This data movement concept was then assigned to the metrology concept of a size unit.".

Considering that data movements is a common feature of many FSM methods there is a quite broad consensus that data movement is a good representation of the concept of the functional size of software. However, each method have different rules for quantifying this concept [8].

As discussed in Chapter 2, we believe definition of functionality is traversable through decomposition levels. Therefore, we perceive the functional requirements as definitions of functionality which can be defined in various abstraction levels. Hence, different levels of functional processes can be defined in corresponding decomposition levels. Functionality allocated to a structural entity in a decomposition level can be broken down into smaller functional processes defined in a lower decomposition level.

In Chapter 2 we suggest that the definition of functionality is defined in a chain of alternating problem and solution domains. That is, a solution (function) poses as a problem for lower levels of decomposition and a problem (expected outcome) will be attained through solutions (functions) in a further lower level of decomposition.

This leads us to the notion that a Base Functional Component for functionality, may exist in both problem and solution domains based on the perspective therefore posing as a common concept in both domains.

The Base Functional Component for Data Movement Point (DMP) measurement method is based on the data movement concept which is commonly present in both domains. We suggest that, through such a method, size information obtained in one domain will be usable in other.

We propose that a measurement method which incorporates decomposition levels into measurement will mitigate the problems defined above.

Estimation of concepts in the problem domain and solution domain will be more accurate if they are based on concepts residing in their respective domains. However, as mentioned before, existing measurement methods are confined in one domain and utilized for normalizing or estimating concepts in the other domain.

We propose a measurement framework consisting of a functional size measurement method, en estimation method and a representation for approximate size.

It is possible to measure software size using the lowest level of decomposition for data movement, automatically. Then, it is possible to scale up the measurement by mapping lower decomposition level objects to objects in higher levels.

As the method is based on a common concept, we believe that, ideally information loss during conversions through problem and solution domains can be prevented. In other words, any such loss will be due to external factors such as human factor or ambiguous definitions but not the ill definition of the measurement method.

We also propose an estimation method and a representation of approximate size in various abstraction levels. This estimation method utilizes size information gathered from higher levels of abstraction of a system and predicts lower levels. In turn, the approximate representation of actual size in higher levels is used as a basis for such an estimation method.

We also present a software tool to perform the measurement method defined.

Below is the way we address the problems defined in the previous section through our solution approach:

Problems of reliability:

Subjective identification of abstract FSM concepts such as Objects of Interest and functional processes comprise the reliability of measurements. We define an atomic

and objective definition of functional process, object and data movement. Through use of a measurement tool, we backfire these concepts of FSM from the source code which is the end point of solution implementation. This totally eliminates the reliability issues.

For higher level measurement results we traverse through higher levels utilizing metadata for the lowest level. Only point of human interpretation is in the generation of this metadata which relates lower level concepts to higher level ones. This mitigates measurement errors as there is less room left for interpretation and renders errors recoverable by fixing the metadata.

Moreover, FSM methods rely on correct and complete systems as they use abstract objects and functional processes. DMP method makes it possible to measure incomplete or imperfect description of systems with minimum impact in the result. As the measurement is finer grained missing information in the system definition will only result in missing measurement information for missing parts instead of botching the whole measurement process.

Problems of granularity:

In the DMP method, information regarding granularity level of measurement is incorporated in the size definition. The lowest level of measurement is atomic and absolute rather that relative as the case with existing methods. This fact renders measurement results to be additive, homomorphic and transitive. That is, system size is the sum of the sizes of its subordinates.

By DMP method, it is possible to measure specifications defined in levels lower than functional user requirements. This makes measurement of components in highly decomposed systems possible independent of other sub systems and components.

DMP method also makes it possible to size software changes that are defined in lower resolution levels than functional user requirements.

Problems in effort estimation:

As mentioned above, studies show that estimation for a concept in a domain relates much better with the size information obtained in that domain and our preliminary studies confirm this [101].

We define an estimation method which rely on the same concepts that the measurement method does. Instead of using conversion factors between two different size measurements that are essentially unrelated. By this approach,

estimations become less prone to gaps between domains and project phases (see section 4.2.1).

Most FSM methods either does not include data manipulations in measurement or just incorporate the size of manipulations as an order of complexity to the overall measurement. As discussed in section 2.2 manipulations defined in a system gradually become movements as decomposition levels deepen. By measuring in lower levels of decomposition, DMP method measures data manipulations which would otherwise be left out in higher levels into measurement. This also increases the accuracy of estimations.

Problems in benchmarking:

Having decomposition level incorporated into measurement results would greatly increase the comparability and normalization of measurement results in the benchmarking datasets, thus, increasing the reliability of any benchmarking activity.

1.3. Research Goals

The primary research goal of this thesis is to provide a comprehensive and systematic measurement framework for sizing software systems which will address the problems stated in 1.1, compliant with principles of metrology, incorporating the solution rationale stated in 1.2.

The framework consists of:

- A measurement method for sizing software systems from detailed design models.
- A method for sizing existing software products by backfiring measureable software models from software code.
- An approximation approach / representation for measurement results including the abstraction level the measurement is performed in.
- An estimation approach for size and effort.

A second goal is to ensure minimum measurement error and inter-measurer variance in measurement results.

A third goal is to propose a common concept to measure problem domain and solution domain concepts so that size information will be traversable between software lifecycles.

1.4. Research Design

The research goals will be satisfied by attaining the following sub goals:

- Studying the ontological background of terms used in software size measurement. Our intention is to define a measurement method compliant with the terminology defined in the ISO International Vocabulary of Terms in Metrology [9] and framework defined by Abran [10] and Habra et al.[11] as well as the ISO/IEC 14143 [12].
- Laying out the concepts related to problem and solution domains for software systems.
- Defining measurand models residing in problem and solution domains. The process model for software measurement proposed by Abran and Jacquet [25] will be used as the basis for the measurement framework to be proposed.
- Designing a measurement method for sizing software systems utilizing detailed design model for system behavior – Sequence Diagrams.
- Designing a procedure to measure existing software products by backfiring behavioral model from source code.
- Extending the measurement procedure for representing different levels of decomposition. It should include additional measurement rules and procedures to define the level of decomposition in which the measurement result is valid.
- Proposing an estimation method that is consistent with the measurement method defined.
- Validating the design of the proposed FSM measurement method to verify whether the measures used satisfy the representation condition of measurement - Theoretical validation.
- Validating the application of the proposed FSM measurement method in order to assess the reliability, correlation with other measurements and correlation with project effort – Empirical Validation. By empirical validation, we refer to the evaluation of the performance of the measurement method through case studies using experimental techniques and statistical data analysis.

1.5. Thesis Outline

In this chapter we discuss the importance of a well defined software measurement method for software size. Point out the problems in the state of art, and summarize our solution approach, research goals and research methodology.

In section 2, we discuss the rationale for separating the problem and solution domain and its impact on software measurement.

In section 3, we summarize the concepts of metrology and how they relate to software measurement theory. We describe the state of the art and existing software measurement methods in the literature with special emphasis on functional software measurement models utilizing software models in problem and solution domain.

In section 4, summarize the methodology and background for defining a proper software measurement method. We define our solution framework consisting of a new measurement method, a model to represent software size incorporating decomposition levels, the measurement tool we developed and an estimation approach to utilize the size measurement method we propose in effort estimation.

In section 5, we define our validation methodology and case studies we performed to validate the measurement method proposed and their results.

In section 6, we summarize the conclusions of the study and discuss further research opportunities.

CHAPTER 2

PROBLEM DOMAIN AND SOLUTION DOMAIN DISTINCTION

The difference between theory and practice equals your ineptitude...

- *Anonymous*

2.1. Mapping the Requirements (Specifications) and Design – Problems

A real world problem, as we humans see it, is a behavior-first domain. Humans perceive a problem or a need by the behavioral aspect first. We first perceive the cause and effect, which are behavioral, then attach those to objects. Solutions however, are object-first, rendering the solution domain an object-first domain.

> **Example:**
>
> **Problem:** We need to **bind** *two pieces of wood* together for them to be able to **withstand** a *load*.
>
> **Solution:** we use a *nail* to **bind** those pieces of wood. We use a *hammer* to **put** the nail through both pieces of wood.

The procedure of solving a problem is, essentially, devising components to attain the desired behavior.

This phenomenon, makes the mapping of problem domain concepts to solution domain concepts arbitrary and non-mathematical.

Similarly in software engineering. The problem is defined as behavior-first and solution as object-first. Migrating from procedural programming techniques to object

oriented techniques in the course of software engineering, also embraces this phenomenon and further clarifies the separation of problem and solution domains.

Problem can be represented as a hierarchy of sub-problems. A higher level problem can be broken down to lower level problems. This phenomenon is defined as Functional Decomposition in engineering.

Similarly, components can also be represented as a hierarchy of sub-components. A larger component can be broken down to smaller lower level components. This phenomenon is defined as Structural Decomposition in engineering.

However there exists no natural relation between these decompositions in two domains. Any mapping in between items in any decomposition level is problem and solution specific and not straightforward. One "item" in the functional decomposition tree in the problem domain may correspond to several "items" in the structural decomposition tree and vice versa. This is the main factor that there is a gap between problem and solution domain which is crossed through use of engineering problem solving.

2.2. What – How

Axiom: Software is a systems that consists of data movements in various levels of granularity. [13] [10]

The question about "how" a problem is solved is essentially the process of breaking down a larger "What" question to smaller "What" to do questions. What question stands for the description of a need or the aim of the system. How question stands for the solution devised for this problem.

One of these transition process in the What-How chain will pose as a boundary between Problem and Solution domains based on the definition of the system at hand.

Example of a Problem- Solution (What-How) chain:

Increase Profit

- …
- Decrease Costs
- Increase Sales
    - …
    - Increase Advertisement
    - Establish an online store
        - …
        - Get a Server
        - Build an e-commerce site
            - Design Web Site
            - Test Web site
            - Develop Web Site
                - Develop Interface
                - Develop Client Side Software
                - Develop Server Side Software

- …
- Develop Database
- Develop Data Abstraction Layer
- Develop Business Layer
    - …
    - Develop Admin Module
    - Develop Stock Module
    - Develop Customer Module
        - …
        - Develop membership functions
        - Develop CRM functions
            - …
            - Develop Customer Class
            - Develop Purchases Class
                - …
                - Develop add purchase method
                - Develop get purchase name method

As one can see from the example chain, the decomposition for a system can happen in any level and may be extended vertically to higher and lower levels. Theoretically this extension is infinite. Higher and higher level problems can be defined as well as lower and lower level solutions.

Engineering is about defining the start and end points for a system. Limiting the highest level with system boundary and lower level with system abstraction principles. In the example, lowest level was the methods of a class, whereas lower levels can be defined for physical bytes in the memory, registers in the memory, bits defining mnemonics of instructions etc.

2.3. Problem Domain and Solution Domain in Software Engineering

Software engineering is no different from other engineering disciplines as far as problem solving paradigm is concerned. Software development is basically developing a solution to a real life problem. Keeping this in mind, we map the software project lifecycle to problem and solution domains. Figure 1 displays the boundaries of problem and solution domains. Positions of work products and software lifecycle activities with respect to these domains are also defined.

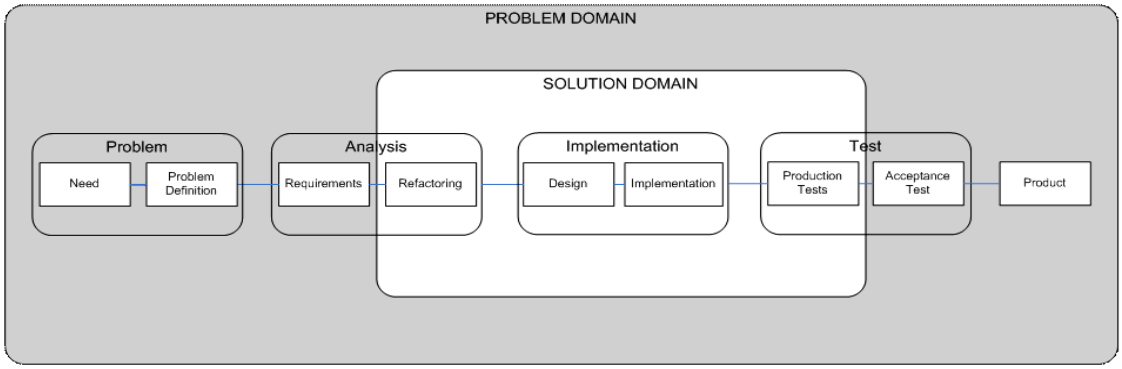

Figure 1 Problem and solution domain borders in a software project lifecycle

Problem domain, by definition, involves the real life need and problem definition. Activities performed to understand the problem such as requirements elicitation and requirements development also lie in the problem domain. Moreover, the validation of the solution, that is, validating whether the solution meets our needs also lie outside the solution box and in problem domain. High-level requirement based testing and related test case generation activities may fall in this category.

Depending on the software lifecycle model used, activity names and content may vary. However, we identify any effort falling within this boundary as problem domain effort.

Solution domain, on the other hand, involves the activities performed towards building a solution to the problem. Typically, these activities include detailed analysis, design, implementation and integration. Implementation typically involves coding and unit test activities.

Similarly, depending on the lifecycle model used, phase and activity names may defer. However, we identify any effort falling in the boundaries between detailed

Concepts of the problem domain	Concepts of the solution domain
Problem statement	
Features	
Functional Requirement	
Systems Specification	Design
	Implementation (Source Code, Model)
	Constructs (Objects, Database Tables, layers)
Price	Technical Specification
Productivity	Performance
	Cost
	Effort
User Test	Unit Test
	Component Test

2.4. Independence of Problem Domain and Solution Domain

In order to prove our suggestion on the separation of problem and solution from a metrics point of view, we conducted two studies [103] [101] with I.Unal. In the first study we investigated the correlation of problem and solution domain sizes with the problem and solution domain effort. In the second study, we conducted a case study where a single set of requirements were to be developed using two different implementation approaches. Both studies supported our suggestions about problem and solution domain separation. A similar but theoretical study by Lavazza and Bianco [14] also demonstrates the independence of problem and solution domain through the Rice Cooker case originally designed by COSMIC [15].

First Study

We analyzed a set of 6 independent software components with a total size of 5036 Cosmic Function Points.

We collected project effort data for analysis, design, implementation and test phases separately. Then, we measured the size of the products in problem and solution domains.

In problem domain, we measured COSMIC function points from the functional requirements. In solution domain, we measured design constructs such as, number of classes, number of operations, number of operation parameters, number of class attributes, number of inter class connections and Source Lines of Code (SLOC). Effort data for different phases of project lifecycle is given in Table 1. Table 2 summarizes the problem and solution domain size measurements for the same components.

Table 1 Efforts for components in person-hours

	Req. Ana.	**Software Design**	**Coding and Unit Tests**	**CSCI Test**	**System Int.**
Prj.1 Comp.1	1354	1192	8799	1715	1116
Prj.1 Comp.2	317	286	1544	76	NA
Prj.1 Comp.3	287	365	959	280	NA
Prj.2 Comp.1	270	492	1281	152	NA
Prj.2 Comp.2	179	328	854	101	NA
Prj.2 Comp.3	90	164	427	51	NA

Table 2 Problem and solution domain measurements for the components

	COSMIC FP	#Classes	#Operations	#Operation Parameters	#Attributes	#Connections	SLOC
Prj.1 Comp.1	1965	2102	8078	4940	17653	1776	234366
Prj.1 Comp.2	482	931	2743	1530	7872	1174	94000
Prj.1 Comp.3	432	209	2390	1216	751	91	24428
Prj.2 Comp.1	638	158	648	2353	2275	74	57831
Prj.2 Comp.2	362	80	720	1078	954	58	48327
Prj.2 Comp.3	1157	160	968	624	162	31	117386

After we collected the data, we investigated following correlations:

- Problem Size (COSMIC) vs. Analysis Effort
- Problem Size (COSMIC) vs. Design Effort
- Problem Size (COSMIC) vs. Implementation (Coding and Unit Test) Effort
- Problem Size (COSMIC) vs. Test Effort
- Design Size (various constructs) vs. Implementation (Coding and Unit Test) Effort
- Design Size (various constructs) vs. Test Effort

Table 3 shows the correlation of design constructs and COSMIC size with implementation and test efforts.

We observed that; solution domain sizes had much higher coefficients of correlation with both implementation and test efforts compared to problem size (FSM).

Table 3 Correlations of problem and solution sizes with effort

Measurement	**Implementation Effort**	**For Test Effort**
# Class	0.9587	0.8867
# Operations	0.9848	0.9553
# Operation Parameters	0.9431	0.9231
# Attributes	0.9531	0.8733
# Connections	0.8773	0.7633
# Parameters + # Attrributes	0.9653	0.8956
# Parameters + # Connectins	0.9677	0.9147
# Para + # Conn. + #Att.	0.9615	0.8878
COSMIC Function Points	0.8055	0.8347

We also observed that problem domain size correlates better with problem domain effort than solution domain effort as suggested. Table 4 presents the correlation coefficients between COSMIC size and effort values. Slightly higher correlation with test effort can be explained with the fact that collected test effort included the effort for test case development activities, which lies in the problem domain as described in section 3.

Table 4 Correlations between COSMIC size and effort values

	Coefficient of correlation for COSMIC Size
Analysis Effort	0.8131
Software Design Effort	0.7777
Implementation Effort	0.8055
Test Effort	0.8347

Figure 2 and Figure 3 show the correlation of number of operations, implementation and test efforts. Number of operations has the best correlation with implementation and test efforts among other design constructs.

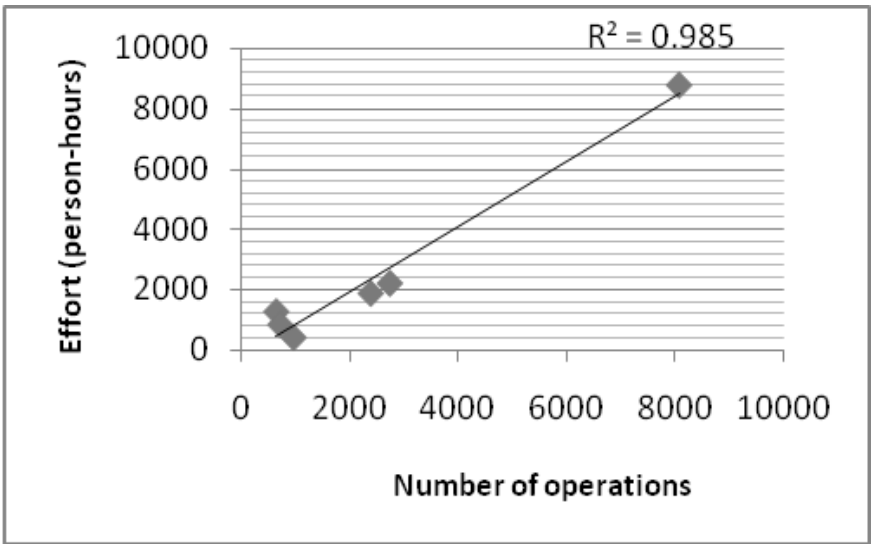

Figure 2 Implementation effort vs. number of operations

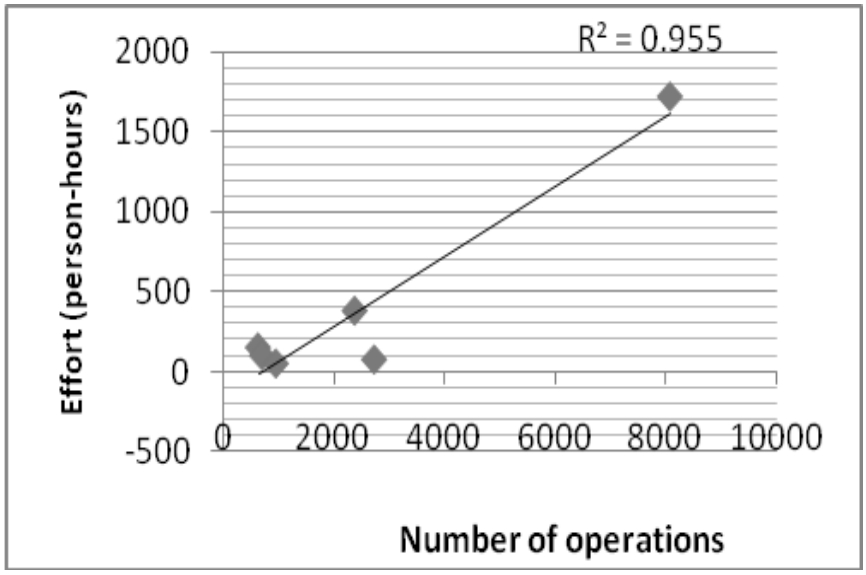

Figure 3 Test effort vs. number of operations

In this study, we observed that solution domain sizes correlated much better with solution domain efforts than problem domain size (FSM) does as suggested.

According to Rubin [91], 20% of the total project effort is used for Analysis. Whereas 21% is used for design, 40% is used for coding and unit test, 10% is used for documentation and 9% is used for deployment. When we map the effort to problem

and solution domains; we can say that 20% of the total project effort lies in problem domain and 80% lies in the solution domain on the average.

Considering the fact that %80 of the total project effort lies in the solution domain, we can say that an estimation method utilizing solution domain concepts would be more accurate in estimating the overall project effort.

Number of operations had the highest correlation coefficient with the project effort. This result also supported our presumptions that data movement in the solution domain would be the best choice for sizing software for the purposes of effort correlation.

Second Study

In this study, we had the opportunity to isolate a single aspect of project characteristics and display its effect on project effort through a comparative study we performed within a software development company.

One of the company's projects involved implementation of a single set of requirements for two different platforms. This gave us the opportunity to conduct a comparative study on two projects which differ only in the implementation. At the end of the study, we demonstrated that implementation technology had a critical effect on project effort, as expected.

First, we compared the projects based on their project characteristics. We chose the attributes defined ISBSG and made the comparison based on these attributes. We expected the projects to be very similar in every characteristic apart from the solution.

We measured the project's functional software sizes using COSMIC FSM method. Having mostly the same requirements, we expected the FSM size of the projects to be close.

Then we collected effort data for each phase in the software life cycle of both projects.

Finally, we compared the effort values for two projects. We investigated how much the effort values differ, when only the development language and operating platform change.

The main functionality required by the two projects was the same. That is, a single set of requirements constituted the majority of the requirements, making most of the software requirements for the two projects identical. On the other hand, when we compare the development effort of the projects we see that Project #1 required 21% more effort than Project #2.

Number of requirements, number of functional user requirements (FUR) and number of common requirements are given in Table 5.

Table 5 COSMIC Size and Effort Values of the Projects

	COSMIC FP	Actual Effort in person-hours
Project #1	1918	17183
Project #2	1965	14177

As all project characteristics other than those regarding implementation technology were constant between the two projects, we can say that the difference in the effort was caused by the difference in the solution domain.

This study demonstrated the disjoint nature of problem domain measurement and solution domain effort.

Henderson [63] and then Desharnais and Abran [34] presented the results of their analysis on the SLOC to IFPUG FP ratio. Both studies concluded that there are large range of variations in SLOC and IFPUG FP sizes.

Rollo [98] conducted an empirical study on 20 applications and concluded that the backfiring functional size to SLOC is greatly inaccurate.

Dekkers and Gunter [48] studied backfiring and conversion of these size measurements based on their fundamentals and reported big variations on the results.

Gencel, Heldal and Lind [16], conducted a study on the relationship between IFPUG and COSMIC sized with Lines of Code and physical size.

They conducted the study in two parts, first on project data obtained from public data set ISBSG 2007 Repository, CD Release 10, then on project data obtained from a single organization.

For the ISBSG Data Set, variation and correlation in SLOC sizes to COSMIC and IFPUG sizes are given in Table 6.

Table 6 Correlation Values for SLOC Size vs. COSMIC and IFPUG sizes in ISBSG Data Set

	SLOC / CFP		SLOC / IFPUG	
	Std. Dev.	R^2	Std.Dev	R^2
Project Set 1	6.04	0.559	-	-
C Language	-	-	125.1	0.26
Visual Basic	-	-	48.0	0.66
SQL	-	-	81.0	0.61

For single organization's data set, variation and correlation in SLOC sizes to COSMIC is given in Table 7.

Table 7 Correlation Between SLOC and COSMIC Sizes in a Single Organization

	SLOC / CFP	
	Std. Dev.	R^2
Comfort & Convenience Type Components	8.2	0.4855
	15.4	0.417

For single organization's data set, variation and correlation in Physical size (Bytes) to COSMIC size is given in Table 8.

Table 8 Correlation Between Physical Size (Byte) and COSMIC Sizes in a Single Organization

	Bytes / CFP	
	Std. Dev.	R^2
Comfort & Convenience Type Components	7.6	0.99
Display & Indication Type Components	21.6	0.992

As one can see from their results, they observed very weak correlation and significant variation for the SLOC/CFP ratios.

However the degree of variation for the ratios Bytes of code per FP was significantly smaller than the SLOC per FP. Moreover, correlation was very high. This again supports our suggestion about problem and solution domains as COSMIC size, which is a problem domain measurement, fails to correlate with SLOC which resides in solution domain and correlates greatly with physical size which resides in problem domain.

Staples et al. [17] also conducted a similar study and their findings were in line with the previous studies stating that there is very weak relationship between CFP and SLOC and concluded that CFP should not be used as a predictor of source code line. Moreover, they found very strong correlations between line counts for the formal specifications and source lines of code. As formal specifications lay in the solution domain, this also strengthens our suggestion stating that problem domain and solution domain measurements correlate better with their problem and solution domain constructs respectively.

2.1. Measurement in PD and SD

As we mentioned before, there exist several measurement methods and different software size definitions. Based on the separation of domains defined above, we categorize these as problem domain and solution domain sizes based on the software artifacts from which they originate. A software size definition is categorized as problem size if the artifact used to measure it is produced in the problem domain. Similarly, a size definition is categorized as solution size if the artifact used to measure it is produced in the solution domain. Any size measured from software requirements is identified as problem size. Similarly, any size measured from, design constructs is identified as solution size.

Some of the commonly used size definitions are categorized as follows:

Problem domain sizes:

- Number of requirements.
- Feature Points.
- Function Point (FP) based sizes. (IFPUG, COSMIC, MARK II etc.)
- Use case points (high level)
- Object Points
- 3D Function Points

Solution Domain sizes:

- Use Case Points (low level)
- Object Points
- Object Oriented Design Constructs (Number of attributes, number of operations, number of connections etc.)
- Lines of Code based sizes. (Source Lines of Code, Logical Lines of Code, Number of instructions etc.)
- Web Objects
- Data Points
- Object Oriented Function Points - OOFP

- Predictive Object Points
- Shepperd & Cartwright Size Measurement

Figure 4 depicts the timing of several measurement methods, based on the availability of work products utilized as measurand models for the methods.

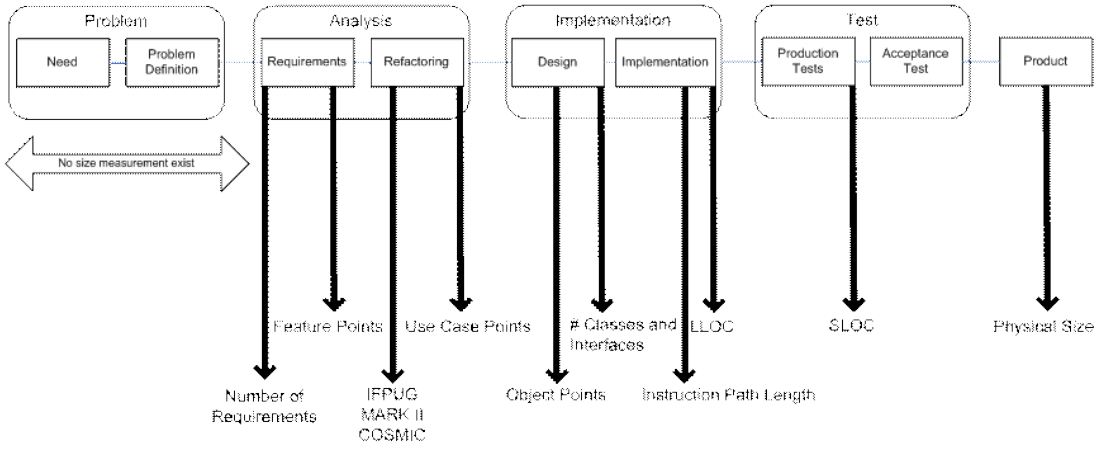

Figure 4 Position of several measurement methods based on the availability of input models

CHAPTER 3

METROLOGY AND SOFTWARE MEASUREMENT

Everything that can be counted does not necessarily count; everything that counts cannot necessarily be counted

- Albert Einstein

3.1. Measurement Theory

Measurement procedure is defined in the International Vocabulary of Basic and General Terms in Metrology [9] as a "*set of operations, described specifically, used in the performance of particular measurements according to a given method of measurement*".

A method of measurement is defined in this vocabulary as a "*logical sequence of operations, described generically, used in the performance of measurements*".

3.2. Size & Effort – Approximation & Estimation

The terms; software size, software estimation, size approximation, size estimation, effort estimation, effort measurement are often confused and used interchangeably.

The software metrics community has generally not defined and differentiated these concepts which is still a problem resulting in confusion in both academics and industry.

Software Size: Is the size of a software product or an intermediate work product to develop the software product. Can be represent in LOC, Function Points, COSMIC Function Points, Bytes, number of methods etc.

Effort: Is the amount of effort spent in a project or a project phase. Represented with man months or man hours. The term project size is sometimes used in confusion to define effort.

Measurement: is the activity to quantify a concept that is already present or happened.

Approximation: Is the activity to give an approximate value for the quantification of a concept that is already present or happened.

Software size measurement: Is the activity to quantify the size of measurement. Some of the measurement methods are SLOC, Function Point analysis, IFPUG, COSMIC, Use Case Points.

Software size approximation: Is the activity to give an approximate value for the size of the measurement. Can represented as an interval or a value with corresponding error margin.

Size Estimation: Is the activity to predict the value of software size that is yet to be created. Can represented as an interval or a value with corresponding error margin.

Effort Measurement: Is the project management activity, to measure the amount of effort spent in an activity, project phase or total project. Time sheets is a common method of effort measurement.

Effort Estimation: Is the activity to predict the effort that is yet to be spent on an activity, project phase or total project in the future. As far as software projects are concerned, cost of a project is most of the time directly related to the effort put in the project. Therefore the term cost estimation is also used to define this activity.

3.2.1. Effort Estimation

Project planning poses the base of project management. Therefore effort estimation is one of the most important activities in project management. However, effort estimation practices are still far from being accurate for software projects [47][60].

The main reason that estimation is difficult is that it has to be done at the beginning of a project. At that point very little is known about the project. The further in time the project goes, the more is known and the better estimations for the remaining work get. Accuracy of estimations can be represented with the percentage of absolute value of deviation of the estimation from actual project effort i.e Magnitude of Relative Error (MRE)

$$\mathrm{MRE_{TE}} = \frac{|Actual\ Total\ \ Effort - Estimated\ Total\ Effort|}{Actual\ Total\ Effort} \qquad \text{(Equation 1)}$$

$$\mathrm{MRE_{RE}} = \frac{|Actual\ Remaining\ \ Effort - Estimated\ Remaining\ Effort|}{Actual\ Remaining\ Effort} \qquad \text{(Equation 2)}$$

One should note that the Actual Total Effort and Actual Remaining Effort are actually hypothetical values at the time of estimation. They can only be known at the end of the project. Therefore these error values can only be calculated only retrospectively for a project.

As a project propagates in the time, attributes of the project and the required software product gets clearer. Estimations for the total project effort in each phase tend to be more accurate.

3.3. Measuring Software Size Based on Software Design Models

Proposal of Bévo et al. (1999)

Bévo et al. [45] associates the concepts of UML version 1.0 and COSMIC-FFP version 2.0. Their approach is based on use cases and class diagrams. Each use case maps to a functional process in COSMIC notation. Actors of a use case are considered as functional users. Scenarios of a use case are transformed into data movements and each class of a class diagram is mapped with a data group. However, triggering events and measurement layers in COSMIC notation are not mapped to any UML concept. Proposed approach was verified with five case studies in [46]. The procedure is applied in a measurement tool named, Metric Xpert.

Proposal of Jenner (2001)

Jenner [73] evaluates the model of Bévo et al. and improves the model by mapping additional UML concepts to COSMIC concepts. Unlike Bévo et al. he (she) maps each functional process to a sequence diagram. Interaction messages in each sequence diagram are mapped to data movements. She suggests usage of swim lanes to represent measurement layers. This procedure is also supported by a measurement tool [72].

Proposal of Diab et al. (2001)

Diab et al. [18] propose a model for the measurement of real time applications' functional size. The model is based on Rational Rose Real Time (RRRT) model. Layers of a real time application are represented by a set actors at the same level of abstraction. Transitions in the application are mapped to functional processes. Actions and messages in the RRRT models' state machine diagram are mapped to data movements. Data groups are represented by actors and protocol classes and data attributes are represented by the attributes and variables of the classes. Proposed approach is supported by a tool named µcRose [19].

Proposal of Poels (2002)

Poels's [89] model which was developed by associating the concepts of COSMIC and the concepts of the business model and the services model of MERODE [14] allows measurement of multilayered applications. The model is proposed for the measurement of management information systems applications. Based on the business model and COSMIC mapping; functional processes corresponds to a set of class methods and data movements are mapped to each of these class methods. Classes of the business model corresponds to data groups.

On the other hand for the services layer, each functional process corresponds to a non-persistent service object of the services model and each data movement is mapped to class methods.

This proposal of Poels has no support of a measurement tool and was only verified theoretically in [20].

Proposal of Nagano et al. (2003)

Unlike Bévo et al. [46], Nagano et al.'s proposal [83] allows measurement of real time applications from xUML [80] concepts. The model utilizes Class, state transition, and collaboration diagrams. The attributes of the class diagrams, message parameters and control signals are considered as candidate data groups. Collaboration diagrams are utilized for the identification of triggering events. Finally, set of data movements in collaboration diagrams correspond to functional processes.

The proposal does not supported with a measurement tool. In addition, it was mentioned that the result of the case study conducted with Rice Cooker case [53] displayed %53 difference from the original measurement result.

Proposal of Azzouz et al. (2004)

Azzouz et al. [41] presents a proposal based on the fundamentals of Bévo's [45] and Jenner's [21] models and develops a tool to automate the functional size measurement process of management information systems projects developed with Rational Unifies Process (RUP) methodology. The model utilizes use cases and use case scenarios in three phases of the development methodology. These phases are business modelling, requirements analysis and design. One advantage of this proposal is that the tool had integration with Rational Rose tool.

Proposal of Condori-Fernández et al. (2004)

Condori-Fernández et al. [22] presents a proposal to measure the functional size of object oriented systems. The proposal works based on the OO-Method requirements

model [88] including functions refinement tree, use case diagrams and sequence diagrams. Use cases and functions of the refinement tree correspond to functional processes. Sequence diagrams' elements corresponds to data groups and data movements. The model does not explain triggering events. Although the model does not have a tool support, it has been verified in [52],[50] and [51].

Proposal of Habela et al. (2005)

Habela et al. [23] presents a mapping of UML version 1.5 and COSMIC FFP version 2.2 in the use case context. The proposal depends on detailed use case definitions and use case diagrams. Use cases are mapped with functional processes and scenario descriptions are mapped with data movements. In the literature there is no such study to describe the verification of the proposal.

Proposal of Levesque et al. (2008)

Levesque et al. [24]develops a model for the measurement of management information systems from use cases and sequence diagrams. In the model, each use case corresponds to functional process and each actor of the use case corresponds to functional user. Sequence diagram elements are mapped to data groups and data movements. In this model data manipulations are also taken into account. Error messages in the sequence diagrams correspond to data manipulations.

Levesque's proposal does not supported with a measurement tool. In addition, it was mentioned that the result of the case study conducted with Rice Cooker case [53] displayed %8 difference from the original measurement result.

Proposal of Marín et al. (2008)

Marín et al.'s [25] proposal allows measurement of object oriented systems developed using OO-Method. OO-Method, being a Model Driven Architecture approach, has a three tier architecture: presentation tier, logic tier, and database tier. Layer concept of COSMIC measurement method is associated with these tiers. Interaction units in presentation tier are associated with functional processes.

On the other hand, the proposal involves three models: the requirements model, the conceptual model and the execution model. The conceptual model is composed of four models: the object model, the functional model, the dynamic model and the presentation model. Classes in the object model are associated with data groups, whereas attributes of the classes are associated with data attributes.

Finally, the proposal has a well defined rule structure, tool support and has been verified using various case studies.

CHAPTER 4

DATA MOVEMENT POINT (DMP) MEASUREMENT METHOD AND CORRESPONDING ESTIMATION APPROACH

For millenia, scientists always try to measure the size of this vast universe. One way to know that is first to find the smallest single thing that constructs this universe. When they get it, the real measurement of universe can be understood for sure.

— Toba Beta

We are suggesting a two way approach to the same continuum of concepts. The measurement method, is a bottom up approach which have its base in the bottommost relevant representation of data movement and consolidated upwards with increasing granularity. Estimation method, however is a top down approach, starting from the topmost representation of data movement and broken down with decreasing granularity.

The optimum level of granularity to collect historical data and, hence, base the estimations on, should be identified per organization and/or domain.

Many software systems are developed in an iterative manner, adding detail as required. In object-oriented systems this is particularly common, and applies to all stages of development from analysis of requirements to final detailed design. [21]

In the first part of this section, we present the framework for software measurement method design as defined by Abran and Habra et al. [8] [11].

Defining measurement activity as a mapping between the real world and the numerical world is not enough as a measurement should involve information about how to measure and how to have a sufficient degree of confidence in the measurement results.

The VIM [9] describes software measurement in three levels: Measurement principle, measurement method and measurement through a measurement procedure (see Figure 5)

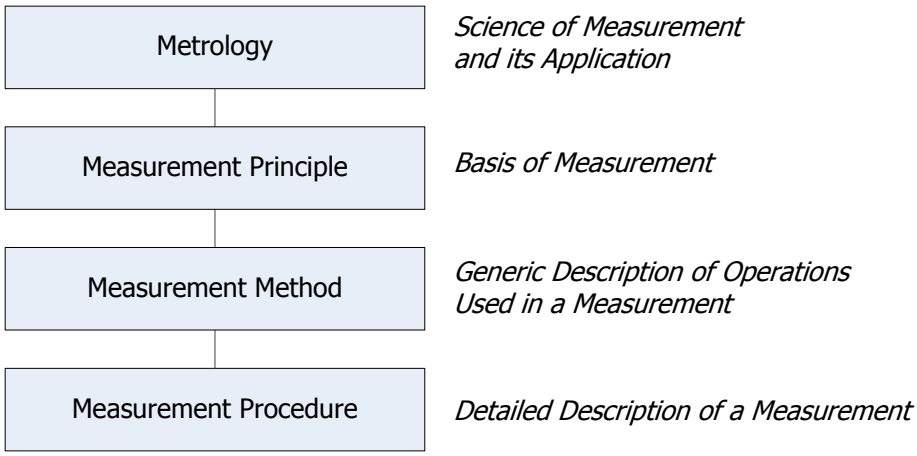

Figure 5. The levels in the measurement foundation in the VIM [9]

Habra et al. defines these steps as [11]**:**

The measurement principle is a precise definition of the entities concerned, and the attribute to be measured. According to the representational approach of measurement, the measurement principle involves a description of the empirical world and the numerical world to which the entities are to be mapped.

> **a**. **The empirical world** can be described through conceptual modeling techniques or through mathematical axioms, or both.
>
> **b. The numerical world** can, in the general case, be any mathematical set, along with the operations performed on it.

It can also be defined through the selection of one scale type (ordinal, interval, or ratio). This also includes the definition of units [9], and other permitted composition operations on the mathematical structure.

2. The measurement method is a description of the mapping that makes it possible to obtain a value for a given entity. It involves some general properties of the mapping (declarative view), along with a collection of assignment rules (operational description).

> **a. Declarative mapping properties** can include a description of other mapping properties, in addition to the homomorphism of the mapping. For instance: a unit axiom (the mandatory association of the number 1 with an entity of the empirical set); or, more generally, an adequate selection of a small finite representative set of elements ranked by domain practitioners.

b. The numerical assignment rules correspond to an operational description of the mapping, i.e. how to map empirical objects to numerical values, and include: identification rules, aggregation rules, procedural modeling of a measurement instrument family, usage rules, etc.

Firstly, the definition of the measurement principle that should embody our knowledge or our understanding of the concept to be measured, that is, according to the vocabulary above, the entity and the attribute under consideration. In other words, this activity gives the precise description of what we are going to measure. The second activity is the definition of a measurement method on the basis of that principle. This activity gives a general description of how to measure. And the third activity is the determination of an operational measurement procedure that is an implementation of the method in a particular context: this third activity gives a detailed description of how to measure.

4.1.1. How to Measure Software

- Step 1: Before measuring, it is necessary to design a measurement method.
- Step 2: The rules of the measurement method are applied to a software or piece of software;
- Step 3: The application of the measurement method rules produce a result.
- Step 4: The measurement result is exploited in a quantitative or qualitative model.

4.1.2. Defining the Measurement Principle

The definition of the measurement principle gives the precise description of what we are going to measure. For software entities (products), the measurement principle involves the model(s) used as a basis on which to describe the entity for which a given attribute is intended to be measured.

The idea is that modeling, as a central notion in software products, should be considered at the same level as scientific principles in other sciences and in engineering. Accordingly, the modeling activity (corresponds clearly to the activity of defining the measurement principle.

The measurement principle is defined in several aspects:

- Context
- Describing the Empirical World: Characterization and Modeling
- Modeling Techniques for Describing the Empirical World
- Mathematical Techniques
- Conceptual Modeling Techniques
- Representative Elements

- Describing the Numerical World: Scale Types and Units

4.1.3. The measurement Method

According to the ISO metrology vocabulary of the VIM [9], a measurement method is defined as a generic description of a logical sequence of operations used in a measurement. Therefore, it should give a first operational definition of the mapping described above, that is, an operational description of how to map a given empirical entity to its corresponding value.

More precisely, these rules define how to find, in practice, the values associated with a particular attribute of an empirical entity. At this level, the description of the method should be given as a set of operations. The declarative properties of the level above (e.g. the measurement principle) characterize the mapping from empirical entities to numerical entities; but, in concrete terms, an operational process, such as counting, calculating, etc.,

In other words, this part corresponds to the design of the operationalisable method according to which the measurement should be achieved. If the measurement method involves counting operations, then it describes the rules that should precisely determine:

- How to distinguish the entities to be counted,
- What should be disregarded,
- How to perform the count, etc.

4.1.4. Measurement Procedure

A measurement method should, in turn, be implemented concretely by a measurement procedure, which describes a measurement according to one or more measurement principles and to a given measurement method. It consists of concrete operations performed by means of measuring instruments and/or practical actions such as selection, counting, calculation, comparison, etc. It is more specific, more detailed, and more closely related to the environment and to the measuring instruments (e.g. tools) than the method, which is more generic.

Determination of an operational measurement procedure, that is, an implementation of the method in a particular context, is not considered as part of the design step, but is carried out with every measurement exercise to ensure both the accuracy of the measurement results and the traceability of the measurement exercise.

According to the ISO metrology vocabulary of the VIM [9], a measurement procedure is defined as a detailed description of a measurement according to one or more measurement principles and to a given measurement method:

- This level of description corresponds to a yet more operational and more practical definition of how to map an empirical entity to its corresponding number.
- For practical purposes, a measurement procedure corresponds to a measurement report which gives a precise implementation of a given method for a specific context or set of contexts.
- Moreover, if the measurement process goes through a measuring device, this level involves a precise description of that instrument, its calibration, and the documentation of its metrological properties, such as accuracy [9].

Therefore, an important aspect of the definition at this level is to precisely describe the context in which the measurement procedure will take place. The context involves various parameters that are worth investigating. At a minimum, the parameters that should be taken into account are:

- the purpose of the measurement process, and
- the constraints under which the measurement will be performed.

Some constraints are related to a particular application of the measurement method (e.g. existing measuring devices, the possibility of experimentation, available representative sets, etc.)

4.2. Measurement Principle

4.2.1. Context

Through the phases of software lifecycle, there are several models (work products) representing the software in that phase. Each of these models can be used as an alternative measurand to measure the software at that phase.

In each phase, information about the software goes through a transformation. The problem in the real life is transformed to a problem definition through requirements elicitation. Problem definition is transformed into requirements through business analysis. Requirements are transformed into specifications through software analysis. Requirements are transformed into design through designer's subjective decisions and choices, design patterns and trends. Design is transformed into implementation (coding or model construction) using development frameworks and methodologies. And implementation is transformed into the physical software product in the computer memory through compilers and connectors.

In practice, each transformation introduces a gap between the previous and next levels of representation of the software. Of course all these gaps are different by nature. They are not equally wide or noteworthy. There exists several practice in the software engineering discipline which aims to minimize these gaps.

Gap 1: Real life problem – Problem Definition: Not every aspect of the real life problem is captured in the requirements elicitation or some aspects are misunderstood. Minimized by better elicitation techniques and customer reviews.

Gap 2: Problem Definition – Requirements: Software Requirements do not cover the whole problem definition or misplaced due to earlier misunderstandings. Minimized by requirements quality reviews and validation.

Gap 3: Requirements – Specifications: Not every requirement is correctly transformed into specifications or irrelevant specifications are introduced. Minimized by traceability and reviews.

Gap 4: Specifications – Design: Physical Software design that is built to meet the specifications can vary greatly, and may involve mistakes. Mistakes can be removed through reviews. Variance is essential.

Gap 5: Design – Implementation: Implementation of design can also vary per individual developer, development environment, programming language, development method, utilized libraries and may also include errors and bugs. Minimized by verification activities such as testing, code reviews.

Gap 6: Implementation – Physical: Physical software output can vary based on the technology and target platform used. No need to minimize as variance is natural and constant based on the technology.

These gaps are cumulative. That is, the total gap between a representation and other is the combination of all gaps in between individual work products. Studies suggest gaps in the earlier phases affect the total gap more than those in the later phases. [47].

In estimation activities, we try to predict concepts related to later phases by using concepts of former phases. That is, we try to predict development effort, which is a concept related to design and implementation, using requirements, which is a concept related to analysis. Similarly, we try to predict physical size, which is a concept related to building, using specifications which is a concept related to software analysis.

We define Gaps 1-2-3-5 and 6 as imperfections in software engineering problem solving process and therefore evitable. We believe all these gaps occur due to inadequacy of performers and non-ideal conditions of the real world. We assume ideally, there exists "one, correct and unique" work product for each relative phase that would eliminate the gap defined.

In summary, we base our approach on the assumption that, hypothetically, in ideal case, there exists:

- Gap 1: One correct and exact definition of the real life problem.
- Gap 2: One correct and minimum set of requirements that describe the problem. Conforming to a requirements standard.
- Gap 3: One correct set of specifications that breaks down the requirements to lower level specifications.
- Gap 5: One best implementation of the design model based on the current state of the art, technology and implementation method used.
- Gap 6: One best and unique software build for the implementation. Through an ideal compiler.

Therefore any gap observed in practice is originated due to non-ideal condition of the real world. Human factors such as experience, bias, error, assumptions and inadequacy of techniques and tools are factors cause these gaps.

However, we perceive Gap 4, the gap between specifications and design is different in nature from these gaps. As devising an engineering solution to a problem involves creativity, background, state of the art and techniques we believe there is no one unique and correct solution to a problem. Engineering solutions are guided by many principles, design patterns and rules. Nevertheless, there is always a subjective, creative element in creating solutions to problems. Apart from this "soft" aspect, changing external conditions, technology and improving techniques and solution approaches changes the way a problem is solved. Therefore we see this gap as essentially inevitable.

The transformation between specifications and design is actually where we define the border between the problem domain and solution domain is.

As mentioned above, there exist several software engineering practices to minimize those evitable gaps in the software lifecycle. Therefore, most of the time the biggest gap resides between specification and design, which is the gap between problem domain and solution domain. From the estimation point of view, an estimation method should propose a way to traverse that gap in order to be able to predict a solution domain concept (development effort) based on a problem domain representation (functional size based on requirements).

However, estimation methods in the literature treat the gaps between different representations of software as black box. They intend to model the transformation or combined transformations in between as a single function and try to define that function based on historical data.

This approach works as long as the gaps between the estimated and estimating representations are mathematical. That is, transformation can be represented as a mapping between two representations. This is, in ideal case, possible for gaps 2, 3, 5 and 6.

Transformation between detailed software design and implementation is mostly straightforward that it can even be automated in both directions. Compiling the implementation (code or model) is purely mathematical.

4.2.2. Describing the Empirical World: Characterization and Modeling

The aim of DMP measurement method is to measure the functional size of software system at a given granularity level and represent the system's size as tuples of two attributes: decomposition level and size. Therefore, the method considers characterization and modeling of the software system in two dimensions: Functional size and structural decomposition.

4.2.2.1. Attribute: Level of Decomposition

The structural model of the software is the main input to determine the decomposition level at which the functional size is measured. The concepts of structural decomposition and functional decomposition are discussed in detail in the section 4.2.3.1.

Base input for the decomposition level is the structural decomposition model of the software.

The sub systems and/or components constituting a system modelled in an arbitrary decomposition level is system specific and cannot be defined specifically, these components are defined generically. The term Structural Component or entity is used for each separate component in a given decomposition level.

4.2.2.2. Attribute: Functional Size

The functional size attribute at each abstraction level is measured based on the "data movement" concept. Data movements are defined as the information conveyed between structural components entities at a given level of decomposition. Total number of data movements in all Functional processes defined in a given level of decomposition constitutes the functional size of software at that decomposition level.

4.2.3. Conceptual Modeling Technique for Describing the Empirical World

Behavioral and Structural Software Models

Instead of a mathematical model, DMP method utilizes conceptual modeling for modeling the empirical world. UML models are used to represent structures, relations among them and behavior of a software system in object oriented software engineering methodology.

Figure 6 displays the structure of UML models. As one can see, the models are separated into two main groups: Structural Diagrams and Behavioral Diagrams. As mentioned above, in DMP measurement method, software size is defined in two components: level of decompositions and functional size, which are measured based on structural model and behavioral model of the software respectively.

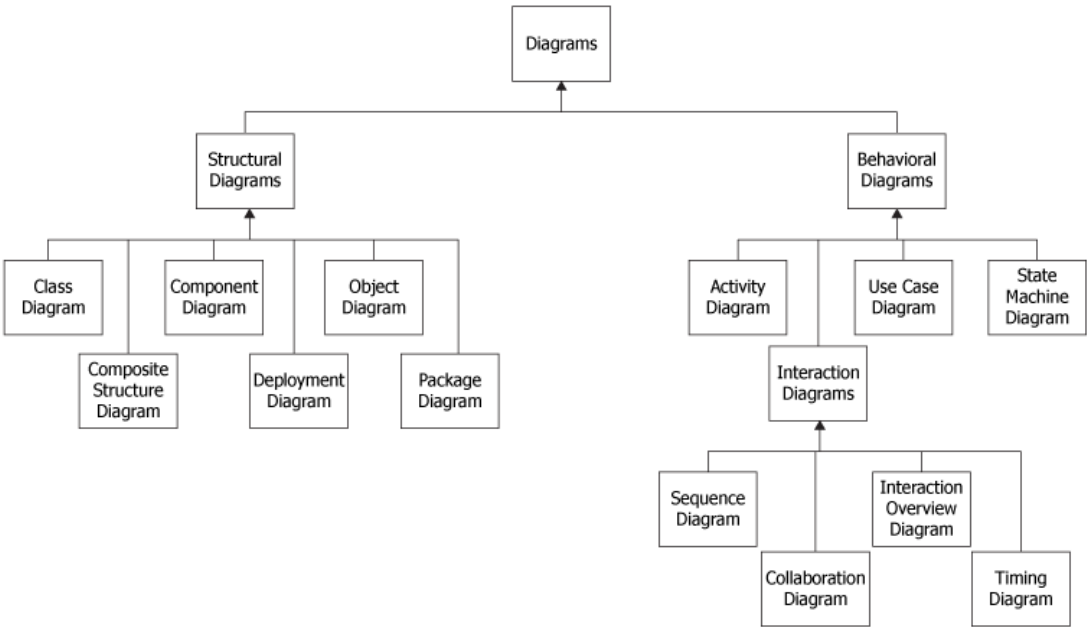

Figure 6 UML 2.0 Superstructure [85]

Conceptual Model for Functional Size

As a conceptual model for functional size attribute, a behavioral model is needed. There are studies in the literature that utilize different behavioral diagrams to measure functional size. (See section 3.3)

For DMP measurement method we needed to use a model which had these properties to meet the needs of the overall measurement approach:

- Enable automated generation of the model by backfiring form the existing software products in the solution domain.
- Can be defined in various decomposition levels.
- Can be used to model concepts both in problem domain and solution domain.
- Give as much information as possible about the software model.

Based on these needs, we chose sequence diagram as the main software model for the empirical world to measure functional size attribute. As:

- It is possible to directly generate code from a detailed sequence diagram and it is also to generate a sequence diagram from the code automatically. There are many CASE tools generate and development environments being used in the industry that is already capable of performing such conversions, also in real time.
- Sequence diagrams can be defined in each composition level as long as the structures and required behavior are defined.
- Sequence diagrams can and are being developed for business level behavior and design level behavior. That is, they can be utilized in both problem and solution domains.
- Sequence diagrams incorporate more information about a system than other UML diagrams. A complete set of sequence diagrams include information about the structures in a system as well as their relations and system behavior.

By counting the data movements depicted in a sequence diagram we can count the number of data movements at a given level of decomposition. Sequence diagrams highlight the data movements between structures, but lack the information about the data manipulations within a structure. As functional size measurement does not essentially measure the internal data manipulations, this abstraction view is a perfect fit to count for data movements.

UML sequence diagrams are useful design tools as they model the dynamic behavior of the system which can be difficult to extract from static diagrams or specifications.

Class/Method level sequence diagrams can be used to generate code directly. Similarly, class/methods level sequence diagrams can be obtained by reverse engineering from the code. This process is defined as "Backfiring". Based on this interchangeability, one can say that backfired lowest level sequence diagrams model software as much as the actual code does.

In the literature, many studies consider higher level sequence diagrams as an input for functional size measurement. [26][27][24][21][28][29][30]

Elements of Sequence Diagrams

Micskei and Waselynck have explained, and discussed the semantics of UML sequence diagrams in detail [104].

Targets

Objects as well as classes can be targets on a sequence diagram, which means that messages can be sent to them. A target is displayed as a rectangle with some text in it. Below the target, its lifeline extends for as long as the target exists. The lifeline is displayed as a vertical dashed line.

Object

The basic notation for an object is:

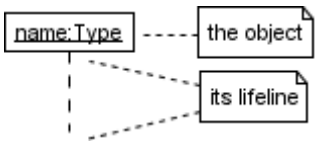

Figure 7: Basic Notation For Object

Where 'name' is the name of the object in the context of the diagram and 'Type' indicates the type of which the object is an instance. Note that the object doesn't have to be a direct instance of Type, a type of which it is an indirect instance is possible too. So 'Type' can be an abstract type as well.

Both name and type are optional, but at least one of them should be present. Some example:

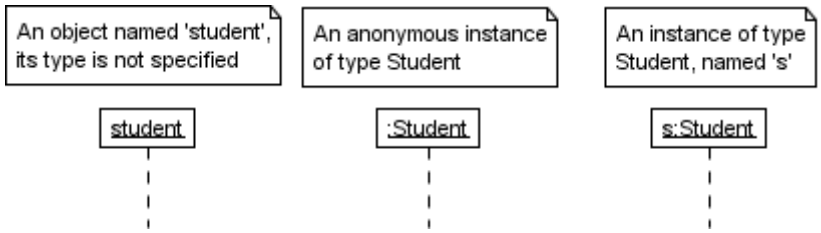

Figure 8: Various Object Representations

As with any UML-element, you can add a stereotype to a target. Some often used stereotypes for objects are «actor», «boundary», «control», «entity» and «database». They can be displayed with icons as well:

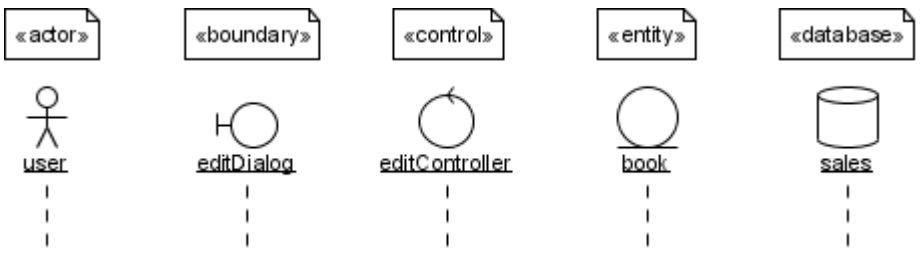

Figure 9: UML Stereotypes

An object should be named only if at least one of the following applies.

- You want to refer to it during the interaction as a message parameter or return value
- You don't mention its type
- There are other anonymous objects of the same type and giving them names is the only way to differentiate them

Try to avoid long but non-descriptive names when you're also specifying the type of the object (e.g. don't use 'aStudent' for an instance of type Student). A shorter name carries the same amount of information and doesn't clutter the diagram (e.g. use 's' instead).

MultiObject

When you want to show how a client interacts with the elements of a collection, you can use a multiobject. Its basic notation is:

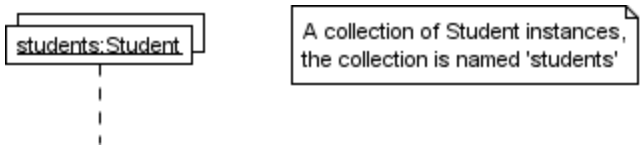

Figure 10: Multi Object Representation

Again, a name and/or type can be specified. Note however that the 'Type' part designates the type of the elements and not the type of the collection itself.

Class

The basic notation for a class is:

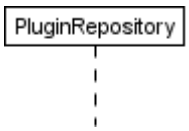

Figure 11: Class Notation

Only class messages (e.g. shared or static methods in some programming languages) can be sent to a class. Note that the text of a class is not underlined, which is how you can distinguish it from an object.

Messages

When a target sends a message to another target, it is shown as an arrow between their lifelines. The arrow originates at the sender and ends at the receiver. Near the arrow, the name and parameters of the message are shown.

Synchronous Message

A synchronous message is used when the sender waits until the receiver has finished processing the message, only then does the caller continue (i.e. a blocking call). Most method calls in object-oriented programming languages are synchronous. A closed and filled arrowhead signifies that the message is sent synchronously.

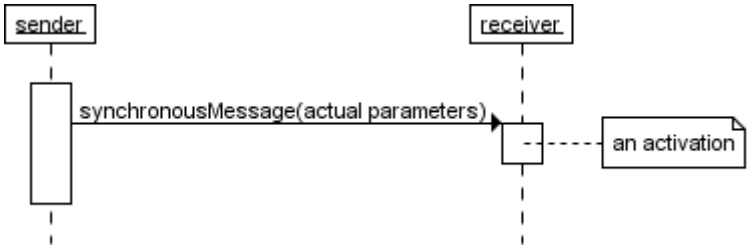

Figure 12: Synchronous Message

The white rectangles on a lifeline are called activations and indicate that an object is responding to a message. It starts when the message is received and ends when the object is done handling the message.

When a messages are used to represent method calls, each activation corresponds to the period during which an activation record for its call is present on the call stack.

If you want to show that the receiver has finished processing the message and returns control to the sender, draw a dashed arrow from receiver to sender. Optionally, a value that the receiver returns to the sender can be placed near the return arrow.

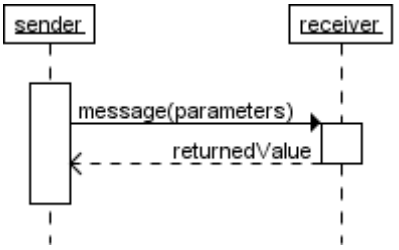

Figure 13: Synchronous Message

If you want your diagrams to be easy to read, only show the return arrow if a value is returned. Otherwise, hide it.

Instantaneous Message

Messages are often considered to be instantaneous, i.e. the time it takes to arrive at the receiver is negligible. For example, an in-process method call. Such messages are drawn as a horizontal arrow.

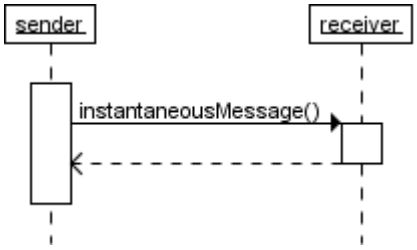

Figure 14: Instantaneous Message

Sometimes however, it takes a considerable amount of time to reach the receiver (relatively speaking of course) . For example, a message across a network. Such a non-instantaneous message is drawn as a slanted arrow.

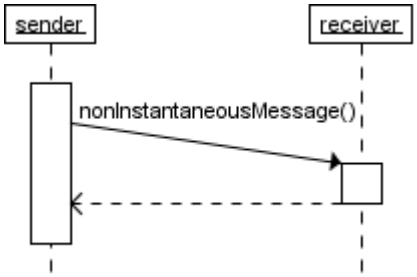

Figure 15: Noninstantaneous Message

You should only use a slanted arrow if you really want to emphasize that a message travels over a relatively slow communication channel (and perhaps want to make a statement about the possible delay). Otherwise, stick with a horizontal arrow.

Found Message

A found message is a message of which the caller is not shown. Depending on the context, this could mean that either the sender is not known, or that it is not important who the sender was. The arrow of a found message originates from a filled circle.

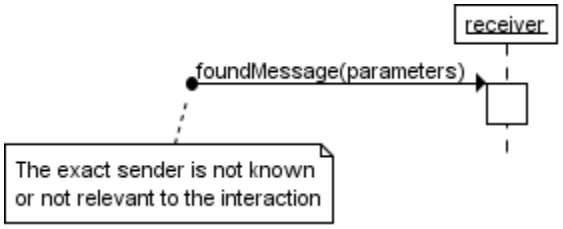

Figure 16: Found Message

Asynchronous messages

With an asynchronous message, the sender does not wait for the receiver to finish processing the message, it continues immediately. Messages sent to a receiver in another process or calls that start a new thread are examples of asynchronous messages. An open arrowhead is used to indicate that a message is sent asynchrously.

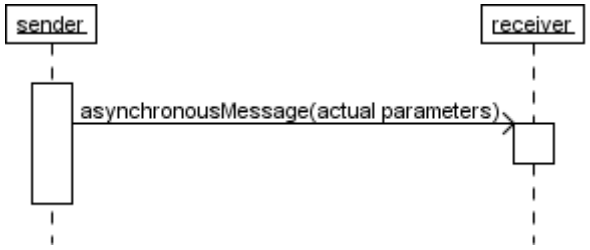

Figure 17: Asynchrounous Message

A small note on the use of asynchronous messages : once the message is received, both sender and receiver are working simultaneously. However, showing two simultaneous flows of control on one diagram is difficult. Usually authors only show one of them, or show one after the other.

Message to self

A message that an object sends itself can be shown as follows :

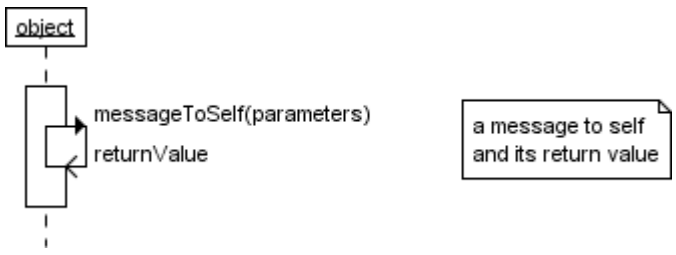

Figure 18: Message To Self

Keep in mind that the purpose of a sequence diagram is to show the interaction between objects, so think twice about every self-message you put on a diagram.

Creation and destruction

Targets that exist at the start of an interaction are placed at the top of the diagram. Any targets that are created during the interaction are placed further down the diagram, at their time of creation.

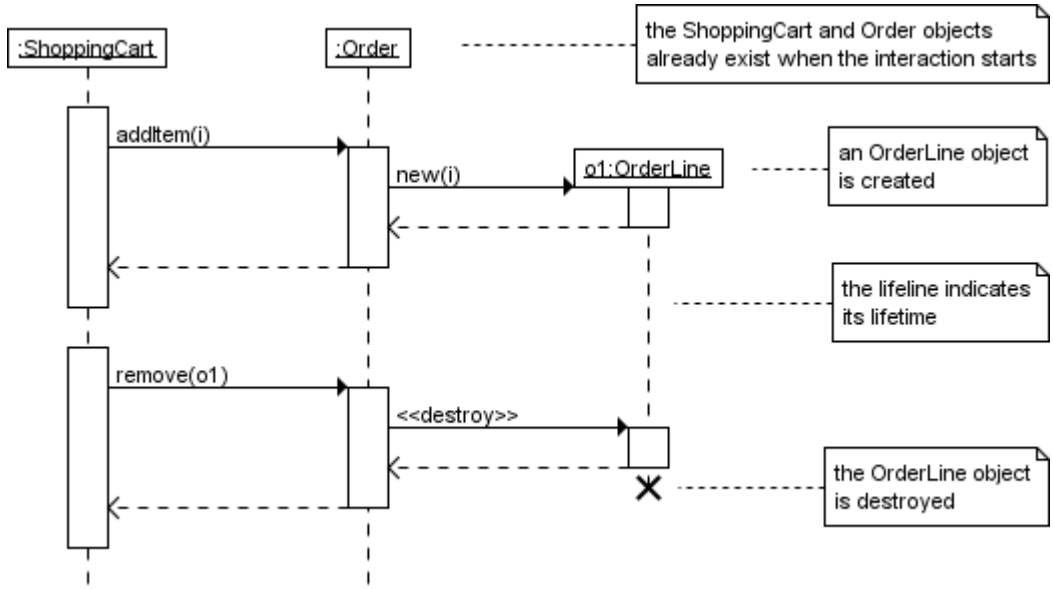

Figure 19: Creation And Destruction

A target's lifeline extends as long as the target exists. If the target is destroyed during the interaction, the lifeline ends at that point in time with a big cross.

Conditional interaction

A message can include a guard, which signifies that the message is only sent if a certain condition is met. The guard is simply that condition between brackets.

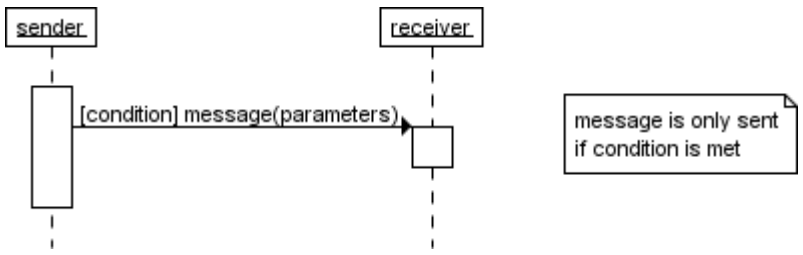

Figure 20: Conditional Interaction

If you want to show that several messages are conditionally sent under the same guard, you'll have to use an 'opt' combined fragment. The combined fragment is shown as a large rectangle with an 'opt' operator plus a guard, and contains all the conditional messages under that guard.

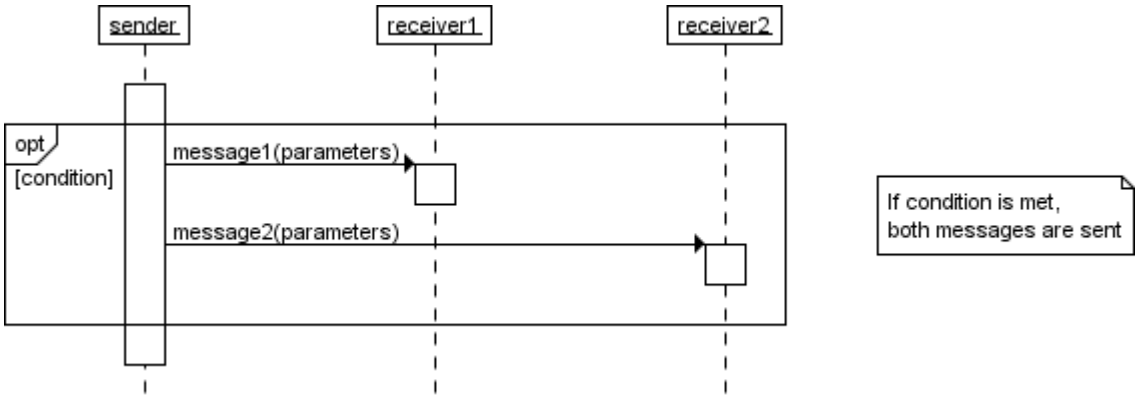

Figure 21: Conditional Interaction

A guarded message or 'opt' combined fragment is somewhat similar to the if-construct in a programming language.

If you want to show several alternative interactions, use an 'alt' combined fragment. The combined fragment contains an operand for each alternative. Each alternative has a guard and contains the interaction that occurs when the condition for that guard is met.

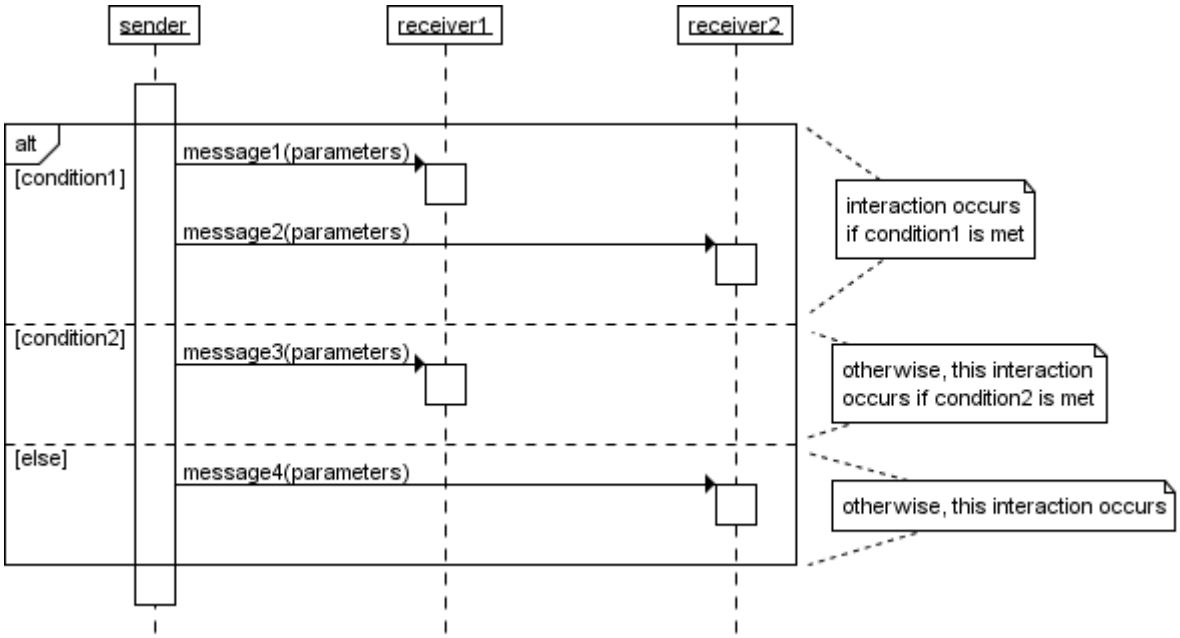

Figure 22: Conditional Interaction

At most one of the operands can occur. An 'alt' combined fragment is similar to nested if-then-else and switch/case constructs in programming languages.

4.2.3.1. Conceptual Model for Decomposition Level

In a decomposition model, each entity of the problem domain, or real world, is represented by an object. An object is an entity which can be of any granularity. Based on the level of decomposition the in which the system is being defined, an object can be a computer system, a person, a machine, a sensor, a software component a class etc.

An object is composed of sub objects and in turn it is a part of a super object itself in the continuum of problem - solution decomposition (See 2.2).

In the continuum of the problem – solution chain, there is a corresponding decomposition model getting decomposed with each step of solutions. As it comes to the problem domain – solution domain border in which software system is defined, the software system may be represented by a single object that interacts with other objects in the real life.

Figure 23 abstractly depicts the decomposition of a system at hand.

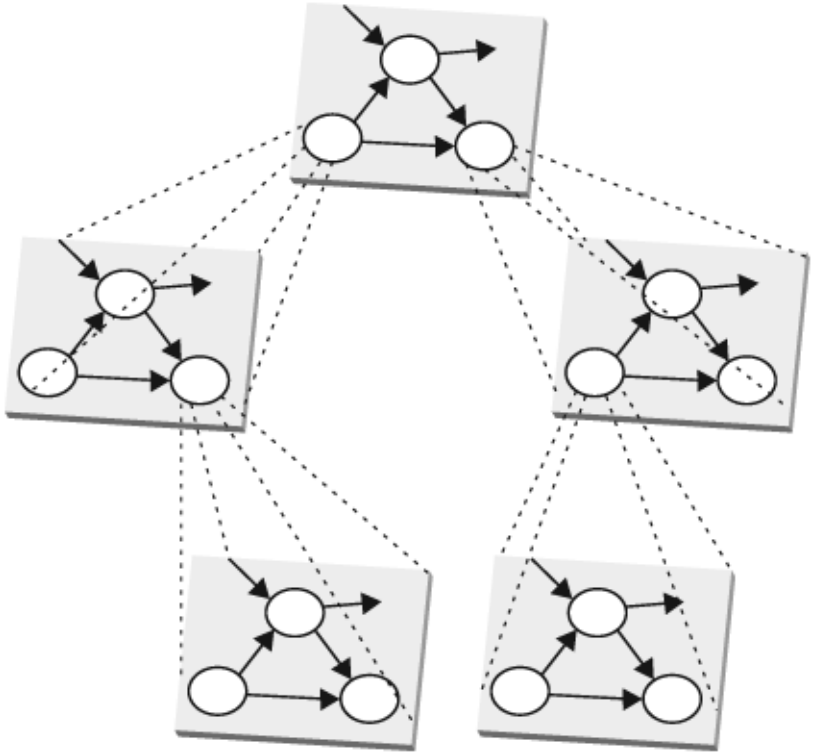

Figure 23. Levels of decomposition (Partitioning)[31]

Definition of Atomic Level of Decomposition

Beginning at this level each object of the software system may be broken down into component objects on a further level of decomposition. This decomposition may continue until objects are completely decomposed into atomic objects.

We define an atomic object in DMP method as an object so low in the decomposition that, sub objects of that object will have data movements which are out of scope of the objectives of the measurement method. The objective of the DMP measurement method is to measure the functional size of the software to use in project management activities such as effort estimation, performance management, productivity management and benchmarking. Therefore the lowest level of decomposition relevant to this objective would be the lowest level of data movement in which actual effort is put in development. In object oriented software development, this level would be the method level. No lower level functionality involving data movements are implemented in object oriented software development.

It is hard to distinguish concepts belonging to different intermediate levels of decomposition from each other. However, lowest level of decomposition (minimum size of granule) can be identified for a software system, from the development point of view.

Lowest level of decomposition from a development point of view would be the code that is actually developed.

The logical level of abstraction lying within the code may differ based on the technology used. However, the development effort and the size of code will be the same from the developer's point of view.

Based on this, the lowest level of decomposition for Data Movements in an object oriented system design would be the calls (and methods) of an object. There would be no lower levels of data movement to be developed.

Definition of Tier

Each decomposition of the system will result in a new Tier of system definition. Each increase in the tier number represents one higher level of decomposition traversed in the description of the system.

Each tier consists of objects communicating with same tier level of objects. One set of objects in a tier can be present in a higher level if the entry and exit point of data movements between them and other objects does not change on their end. That is, certain object may belong to more than one tier at the same time.

4.2.4. Representative Elements

Representative Elements for Functional Size

There is a wide consensus that, data movements are the basic unit of measurement for functional size. Abran states this as;

The key concept of functionality at the highest level of commonality that is present in all software was identified as the 'data movement'. This data movement concept was then assigned to the metrology concept of a size unit. [8]

In DMP measurement method, sequence diagram elements which represent the data movements are taken into account and other elements are used for supporting information.

Objects or classes are structures that does not have any representative value for data movement and hence, functional size.

Conditional interactions do not impact the functional size as a functionality exist either the path it resides in is chosen during the execution of a sequence diagram or not. Alternative execution paths are discussed in detail in Measurement Method Manual [32].

Similarly, repetitive interactions do not impact the functional size as the measurement is done on the implementation but not the execution of the software. Moreover, number of repetitions will change during each execution. Repetitive data movements are discussed in detail in COSMIC Measurement Method Manual [32].

Representative Elements for Decomposition Level

The tier number of the object triggering the functional process defines the tier of the functional process. Another definition for the tier of the functional process would be the maximum tier of the objects defined in the sequence diagram depicting that functional process. This property for tier number is a natural result of the definition of tier stating that, no object in a tier can request a service from an object in higher tier.

The tier of measurement is the tier of the system definition being measured. Tier numbers of each functional process within a system definition should be the same for a measurement to be represented by that tier number.

4.2.5. Describing the Numerical World: Scale Types and Units

The Unit and Scale for Functional Measurement

The functional size for a sequence diagram is given as the number of data movements in the diagram. That is, the size of functional process in a given decomposition level is in absolute scale mathematically.

The Unit and Scale for Functional Decomposition

The functional decomposition level in DMP method is defined as the tier number of a system definition and hence the tier level of measurement. This

Decomposition level in DMP method is defined as tiers. The method level decomposition is defined as universal Tier 0 for every measurement. Then each consolidated view of decomposition is defined as Tier 1, Tier 2, Tier 3 etc. Tier number is defined based on specific system definition. It is a definitive value and does not infer any ratio in scale, it is a measurement in ordinal scale. The rules of defining the tiers is given in section 4.2.3.1.

Theoretically the maximum level of abstraction that is the Tier Number for the topmost level of decomposition of a system is infinite. However, in reality, behavior of software systems tend to be defined a couple of tiers higher than the methods.

4.3. Measurement Method

The measurement method is defined on the basis of that principle. It is a generic operational description, i.e. a description of a logical sequence of operations, of the way to perform a measurement activity (that is, to move from the attribute of an entity to be measured to the number representing the measurement result). This activity gives a general description of how we are going to measure.

4.3.1. A Mathematical View of the Measurement Method: the Mapping

Etalon for Functional Size

Abran states that,

"The key concept of functionality at the highest level of commonality that is present in all software was identified as the 'data movement'. This data movement concept was then assigned to the metrology concept of a size unit. [8]"

The basic data movement in DMP measurement method is defined as the data movement between classes in the sequence diagram of a method diagram (atomic

functional process). The data movement within a sequence is atomic by definition and cannot be divided further down into other movements. This constitutes an etalon for the functional size whose measurement value is 1.

Reference for Decomposition Level

As decomposition level attribute for measurements is defined to identify the abstraction level corresponding to the functional size attribute, the base unit for decomposition level is the Tier number corresponding to the etalon for functional size. The etalon for functional size is defined as the atomic level data movement represented by the data movement between two classes in the sequence diagram of a method (atomic functional process). Therefore, the base unit of decomposition level is the level of a single method within a single class.

It is important to note that the decomposition level is in ordinal scale. The base unit of decomposition level is defined as Tier 0. Tier 0 is the level of decomposition in which further level of decomposition will be irrelevant of the measurement objectives of DMP method.

4.3.2. An Operational View of the Measurement Method

As we measure the size of the software using data movements as the etalon or base unit, we need a model that represents the data movements in the software. We also need this representation of data movements in any abstraction level chosen. Sequence diagrams are the best model to represent data models in various levels. Moreover, they don't just define the number of data movements within a piece of software but they allocate data movement into respective functional processes therefore representing the usage of data movements and not just the total number of movements.

In DMP measurement method, sequence diagram elements which represent the data movements are taken into account and other elements are used for supporting information.

Sequence Diagram elements that represent data movement are:

- Synchronous message
- Asynchronous message
- Creation message
- Destruction message
- Self message
- Found message

4.4. Measurement Procedure

4.4.1. Mapping Phase

Mapping phase in DMP method corresponds to mapping the scenarios to decomposition levels, that is mapping sequence diagrams to tiers.

A specification begins by identifying the entities in the problem domain and their interrelationships and continue further by detailing the functions performed by and the internal state of each object.

The next step would be to identify which objects could allow decomposition and the layers of abstraction in each decomposition level.

A major advantage object oriented development and UML modeling is that, solution domain entities can be defines in a direct and natural correspondence with the real world, since problem domain entities are extracted directly into the model without any intermediate buffer such as traditional data flow diagrams [94].

Identifying Scenarios

Based on the decomposition level (Tier Value) for the measurement, definition of the functional processes and their triggering entries change.

1. The triggering entry of the functional process must be visible in the system model defined at this level.
2. The structural entity receiving the triggering entry must be visible in the system model defined at this level.
3. The output of the process (Exit or Write) must be visible in the system model defined at this level.

Note that, functional process in higher tiers, will also be preset in lower tiers as their triggering entries will also be present in lower tiers. That is, certain functional processes will have different sizes in different tiers. Typically increasing by the decreasing tier number.

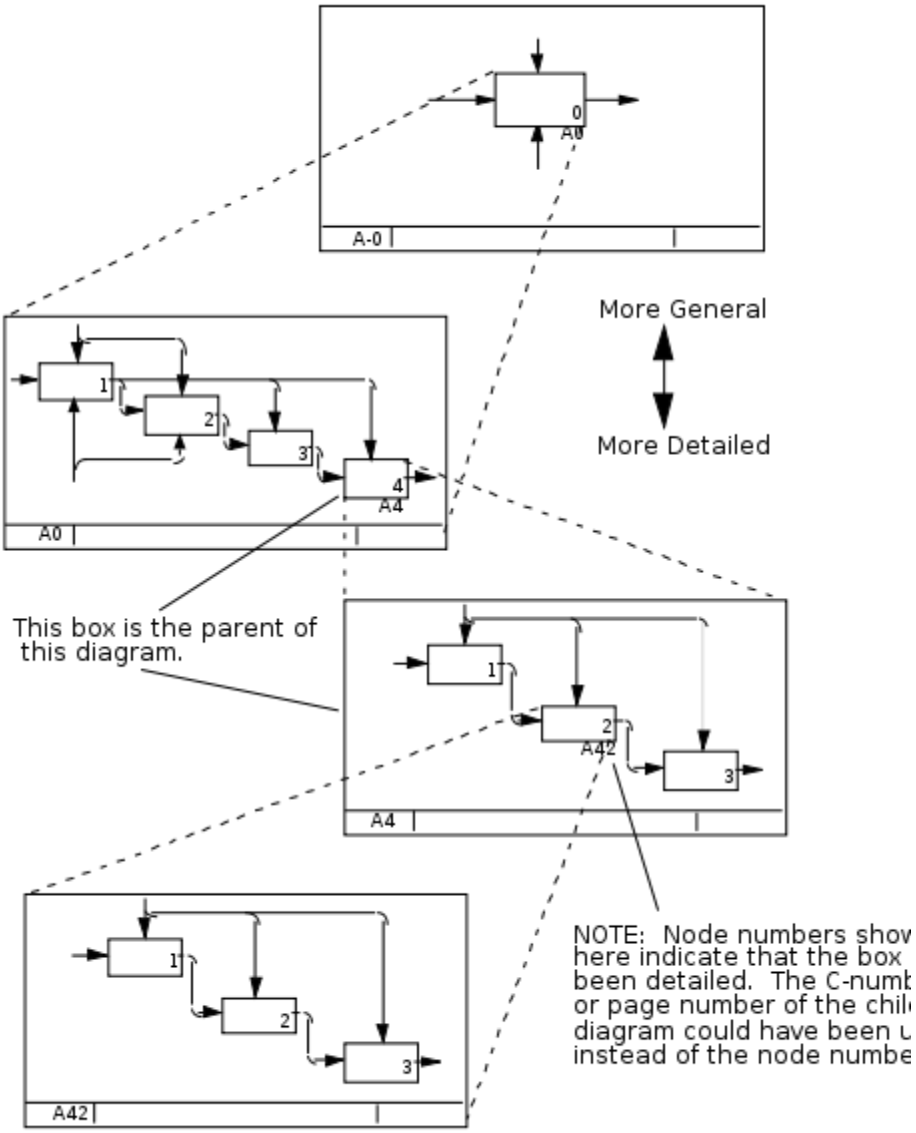

Figure 24 IDEF 0 – Decomposition Structure [67]

Identifying Objects

There exist several rules to check whether a structural entity belongs to a Tier.

- The tier of a structural entity is the level of decomposition it has over the class/method level.
- If there exists no more super entities for an object, that tier is considered the maximum Tier for that object. For further tiers the object is considered to exist in every higher than its maximum tier level.

Identifying Atomic Data Movements

Within atomic level of functional processes, the data movements are represented as method calls between structural entities (objects). This call is not an abstraction or a superstructure but the actual developed method call during the implementation. In other words, this call must be able to be represented as a single code instruction.

Identifying Tiers

The lowest tier level in DMP method is 0. Tier 0 corresponds to the decomposition level in which all communication between structural entities (namely objects, in this level) is carried on with atomic data movements, a single method call.

4.4.2. The Measurement Phase

Once the measurement tier, functional process and objects participating in the process is identified. The sequence diagram for the process is drawn.

In the sequence diagram of the functional process, these calls between objects correspond to data movements:

- Synchronous message
- Asynchronous message
- Creation message
- Destruction message
- Self message
- Found message

4.4.3. Applying the Measurement Function

The size of a functional process is found by counting the total number of data movements (calls) within its sequence diagram.

4.4.4. Aggregating Measurement Results

Calculating the size of a functional process

Rule: The size of a functional process for a tier is the total by counting the total number of data movements (calls) within its sequence diagram in that tier.

Calculating the size of software component or product

Rule: The size of a software component or a software product for a tier is calculated by summing of all its functional process sizes in that tier.

Calculating the size of developed software:

In order to calculate the size of developed software (with the purpose of effort estimation), re-used or COTS components should be identified as a single structural entity in the models.

Calculating the size of delivered software:

In order to calculate the size of developed software (with the purpose of price estimation), re-used or COTS components should also be decomposed and modeled in the required decomposition level so that their sizes will be incorporated to the total measurement.

4.4.5. Measuring Changes in Software

While measuring changes, the tier in which the change is defined should be identified.

4.5. Measurement Tool

Software project evolve in the course of a project and some are developed entirely in an iterative manner. Changing requirements, additional features, size of iterations will all change the size of the software and hence repetition of measurement activities. In that case, automated counting of software size is essential if the size information is utilized in any management activity in a project as manual measurement of any kind of software measurement method is somewhat labor intensive, let alone sizing with design artifacts.

As one may deduct from the DMP measurement method definition, it would be unrealistic to assume that atomic level functional size measurement, which is performed in method level, can be performed manually. Large systems can have hundreds of classes containing thousands of methods. Generating a sequence diagram for each and counting the data movements would be impossible without a tool which will traverse the source code for methods, generate sequence diagrams for each and count the data movements.

We have developed a measurement tool specifically designed to perform this task. The tool was developed as a graduate project in METU Informatics Institute, Software Management graduate program by Yalın Meriç with the co-supervision of Erdir Ungan and Onur Demirörs [33]. The tool was named Sequence Diagram Metric Collector (SDMC).

DMP measurement method utilize sequence diagrams and class information for measurement. In order to automated and communicate UML diagrams, Object Management Group (OMG) has developed XMI which is an XML standard to formalize

UML data and provide a method to exchange metadata information between different systems. XMI standard can be used for any metadata whose metamodel can be expressed in meta-object facility (MOF). Therefore, we developed the SDMC tool to generate and interpret XMI files defining sequence diagrams.

4.5.1. SDMC Requirements

The tool was mainly supposed to count each data movement that took place in each sequence diagram. In order to do this, the tool should have traversed the source code file(s) and generate sequence diagrams for each method.

Generation of Sequence Diagrams

1. The tool shall identify components from the source code.
2. The tool shall identify classes from the source code.
3. The tool shall identify methods from the source code.
4. The tool shall record components, classes and methods in a tree structure.
5. The tool shall generate sequence diagrams for each method identified.
6. The tool shall generate the XMI file(s) for generated sequence diagrams.

Counting for Data Movements

7. The tool shall count the calls inbetween two classes in a sequence diagram and record the class and method at both ends of the call.
8. The tool shall count the asynchronous messages inbetween two classes in a sequence diagram and record the class and method at both ends of the message.
9. The tool shall count the synchronous messages inbetween two classes in a sequence diagram and record the class and method at both ends of the message.
10. The tool shall count and record the instantaneous messages inbetween two classes in a sequence diagram and record the class and method at both ends of the message.
11. The tool shall count and record self messages in a class and record the class and method.
12. The tool shall count and record creation messages inbetween two classes in a sequence diagram and record the class and method at both ends of the message.
13. The tool shall count and record the destruction messages inbetween two classes in a sequence diagram and record the class and method at both ends of the message.
14. The tool shall count the conditions and alternatives in a call in a sequence diagram and record it separately.

15. The tool shall count the loops in a call in a sequence diagram and record it separately.

Generating Measurement Data

16. Based on the data movement data, the tool shall enable consolidation of data, such as:
17. Data movements in between certain components.
18. Data movements in between user defined clusters of objects.

Non functional

19. The tool shall take code file(s) as input, rather than connecting to an IDE environment in order to increase portability.
20. The tool shall be able measure the most popular object oriented programming languages. Such as Java, C# and VB.net.
21. The tool shall be able to measure XMI files created outside the tool and perform the measurement as stated above.
22. The tool shall record the data movement data in a database on which it will be possible to run queries based on component, class, method and call type.

4.5.2. SDMC Solution

In the industry there exist a vast number of CASE tools that help software engineers create, update, manage and share their UML designs. Detailed UML may even be used for automatic code creation. In other words, these tools create all interfaces, classes, methods and attributes based on UML diagrams and generate code in most popular programming languages such as C#, C++, Java and Visual Basic.

Most of the tools also have reverse engineering capabilities alongside code generation. This cycle, generating code from design and build design information from code is defined as round-trip code engineering. After generating the code framework based on UML models, these tools also provide the ability to bidirectional update code and UML diagrams. Whenever a code is updated all attached UML diagrams are updated automatically, and vice versa. Roundtrip engineering support is not common for every UML diagram but almost all popular UML tools provide automatic generation of class and sequence diagrams.

UML modeling tools not only provide automatic UML diagram generation from source codes but also they provide XMI export feature to let engineers share their work among tools and platforms.

The idea was to develop a software tool that imports an XMI file which includes detailed information for sequence diagrams of a software artifact. However, as we

investigated XMI files generated by different tools we noticed that the industry has not reached to a consensus on XMI standards for UML diagrams as XMI files generated by different tools had different structures. OMG has published a diagram definition document [85] to be used for UML diagram interchange but it seems to lack the details needed to exchange UML diagram data completely and accurately.

4.5.2.1. Sequence Diagram and XMI generation

UModel tool from Altova was chosen as the sequence diagram generator and XMI exporter. Altova UModel 2013 is a product developed by a company named Altova and is part of a large collection of CASE tools. UModel was picked as it supported round-trip code engineering and provided an API for automation purposes. Also the tool provided XMI export feature for sequence diagrams. However, the tool did not had a feature to automatically generate sequence diagrams for each method. So the tool had to perform several steps as in a script.

The automated XMI generation process was as below: [33]

1. Select programming language.
2. Select folder where source code files are present.
3. Create a new UModel project.
4. Import source codes under selected folder to UModel project.
5. Automatically generate all sequence diagrams for imported source codes.
6. Save project file to a uniquely named subfolder.
7. Create and save XMI file under above folder.
8. Quit UModel application.

The hierarchical structure of XMI file generated by Altova UModel 2013 was as below:

1. Documentation
    a. Contact
    b. Exporter
    c. ExporterVersion
2. Extension
    a. OpenDiagrams
        i. OpenDiagramEntry
    b. Diagrams
        i. RootElement
            1. guiRootDiagram
                a. guiDiagramGuiLink
                b. guiDiagramLayer
3. Model
    a. packagedElement
        i. packagedElement

1. packagedElement
    a. packagedElement
        i. ownedAttribute
        ii. lifeline
        iii. fragment
        iv. message
    b. ownedAttribute
    c. ownedOperation
        i. ownedParameter
    d. lifeline

ii. profileApplication

After the XMI file was generated, SDMC collected data movement data from each sequence diagram. The basic steps to obtain a set of metrics to be used in size measurement based on sequence diagrams saved as XMI files are as follows:

1. Read XMI file.
2. Collect classes and methods.
3. Collect metrics.
    a. Count synchronous calls between two classes.
    b. Count asynchronous calls between two classes.
    c. Count returned messages.
    d. Count combined fragments (opt, loop, alt).
    e. Count create calls.
4. Save obtained metrics to a database.

After examining the data in generated XMI file the algorithm for collecting metrics was developed and implemented as below:

1. Read XML contents of XMI file generated by Altova UModel.
2. Convert XML content to an object using XML deserialization feature of .NET framework.
3. Collect class and method list in first pass.
    a. Traverse through diagram collection under extension section.
    b. Get related class and method.
        i. Traverse through third and fourth levels of package tree under model namespace.
        ii. If type of package is class then traverse its owned operations collection to find related method by the unique operation id obtained from diagram attributes.
    c. Add obtained class and method to a local list of class entities.
4. Collect metrics in second pass.
    a. Travers through diagram collection under extension section.

b. Get related class and method.
    i. Traverse through third and fourth levels of package tree under model namespace.
    ii. If type of package is class then traverse its owned operations collection to find related method by the unique operation id obtained from diagram attributes.
c. Get metrics related to obtained method.
    i. Traverse through third and fourth levels of package tree under model namespace to find related class using unique package id obtained from diagram attributes.
    ii. Traverse through fragments under located package to collect metrics.
        1. If fragment type is combined fragment then add a combined fragment to details collection under related measurement object. Increase measurement object's number of combined fragments variable by one.
        2. If fragment type is interaction use then obtain related message entity using actual gate attributes of fragment. If message entity's signature attribute points to an operation then obtain that method and class and add two metric details, one for sync call out, one for sync message in movements, to details collection under related measurements object. Increase measurement object's number of sync calls out and number of sync messages in variables by one.
        3. If fragment type is message occurrence specification then obtain related lifeline using fragments covered attribute. If obtained lifeline is the main lifeline then get and process related message using fragments message attribute.
            a. If message type is create message then add a create message to details collection under related measurement object. Increase measurement object's number of create messages variable by one.
            b. If message type is reply then then add a sync message in to details collection under related measurement object. Increase measurement object's number of sync messages in variable by one.
            c. If message type is not defined then add a sync call out to details collection under related measurement object. Increase measurement

object's number of sync calls out variable by one.

d. Add metrics to a local metrics list.

5. Save collected metrics.
   a. Traverse through collected components list and add each component to database.
   b. Traverse through collected classes collection and add each class and its methods to database.
   c. Set time value for all collected metrics to current time.
   d. Add measurement collection to database.

Figure 25 is a screen shot of the SDMC tool.

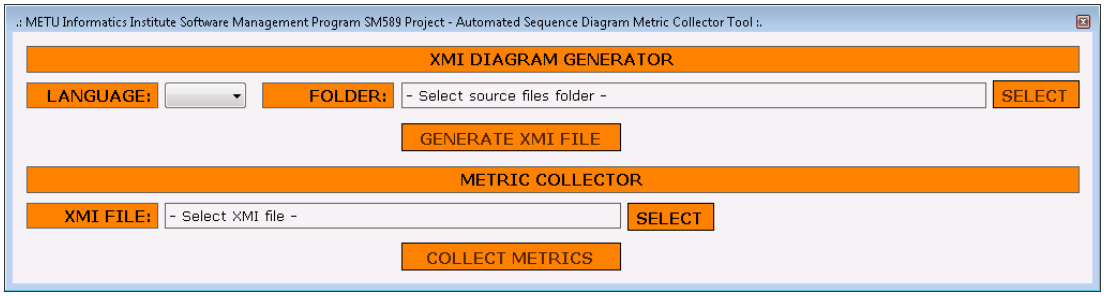

Figure 25 SDMC Screen Shot

4.5.2.2. SDMC Features

1. SDMC gets folders with source code files in it so that, the measurement can be performed independent of the framework or IDE utilized.
2. SDMC can interpret most popular object oriented programming languages such as:

Programming Language	Versions
Java	1.4, 5, 6
C#	1.2, 2, 3, 4
Visual Basic	7.1, 8, 9

4.5.3. Consolidation of Data Movement Counts

As developed in the SDMC only counted data movements in each sequence diagram. In order to obtain a well-structured measurement result, this data was to be consolidated and interpreted. In order to consolidate the data, several queries were designed.

For the base level – Tier 0- measurement, the data should have been queried based on measurement date, project and component.

For the component level, data movements between methods should be grouped by the classes in the components. So that, only data movements coming out and in of the components are counted.

Similarly, for user defined class clusters, which form the layers, design level components and interfaces the measurer defines, the data movements should be grouped by the groups.

These queries were possible as SDMC recorded data on a data base. The structure of the database is given in Figure 26.

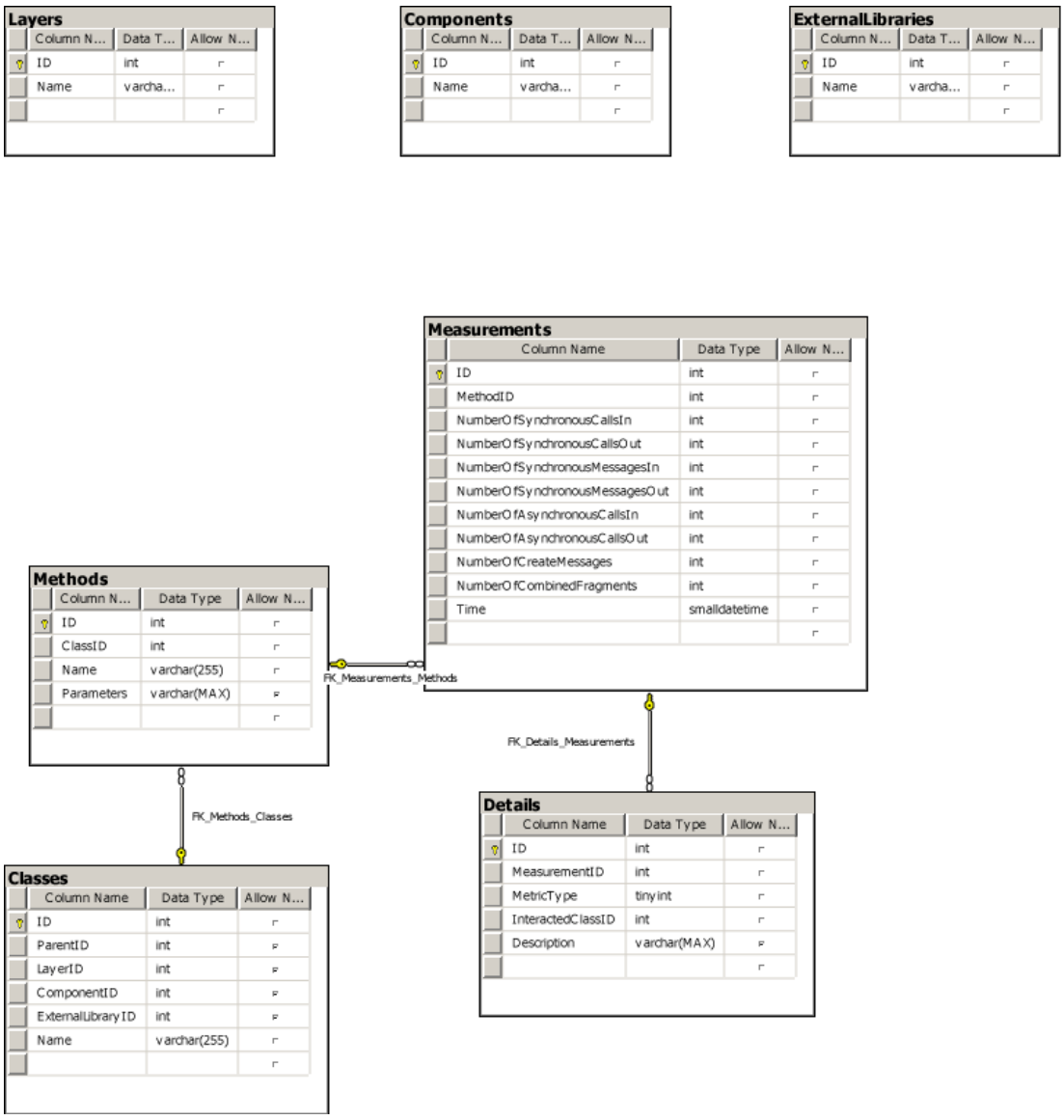

Figure 26 The structure of the SDMC database

In the database, the "Layers" table correspond to the custom defined structures defining higher levels of decomposition and making higher tier measurements possible.

4.6. Estimation Approach

As mentioned before nearly all of the problem domain measurements are based on Alan Albrecht's Function Point measurement [39]. It is based on two assumptions:

- The complexity and size of a software system are major determinants of the length of the development process.
- The complexity and size of a software system can be derived by examining and counting the data complexity and volume.

Functional size measurement methods are defined to be independent of the implementation technology, programming language, implementation environment and methodology. This property is boasted as the main reason for these methods to be a better size input for effort estimation compared to LOC, design sizes or any other solution based measurement. However, it has been shown many times that [16] [103] solution based sizes such as LOC correlates better with development effort. Therefore, being independent from the solution domain concepts may pose a better ground for many purposes such as productivity measurement, pricing etc. but not for effort estimation.

Moreover Problem Domain measures, ignore non functional requirements such as quality, security and performance requirements. They are also not suitable for systems with complex algorithmic processing by definition.

Most of the effort estimation methods utilize historical data to build an estimation function. Earlier methods used historical data to convert Function Points to Lines of Code and then use estimation methods which use LOC as an input.

Later methods utilized historical data to build an estimation function based directly on functional size. In the literature, several methods such as curve fitting, multipliers and neural networks are used to build an estimation function to be used to extrapolate project effort based on functional size for the future projects. Detailed information about various estimation methods are given in section 3.2.

We suggest that estimates should be defined based on the detail of information about the system. Therefore, system decomposition level and functional size should be the major inputs defining an estimation.

As the information we have about a software product and the corresponding project increase, the accuracy of the estimates about remaining effort increase. Beohem [47]

suggest that there that cost estimates will be within the boundaries represented by the two converging exponential curves. Figure 27 shows a plot of the accuracy of software project estimates as a function of the software life cycle phase.

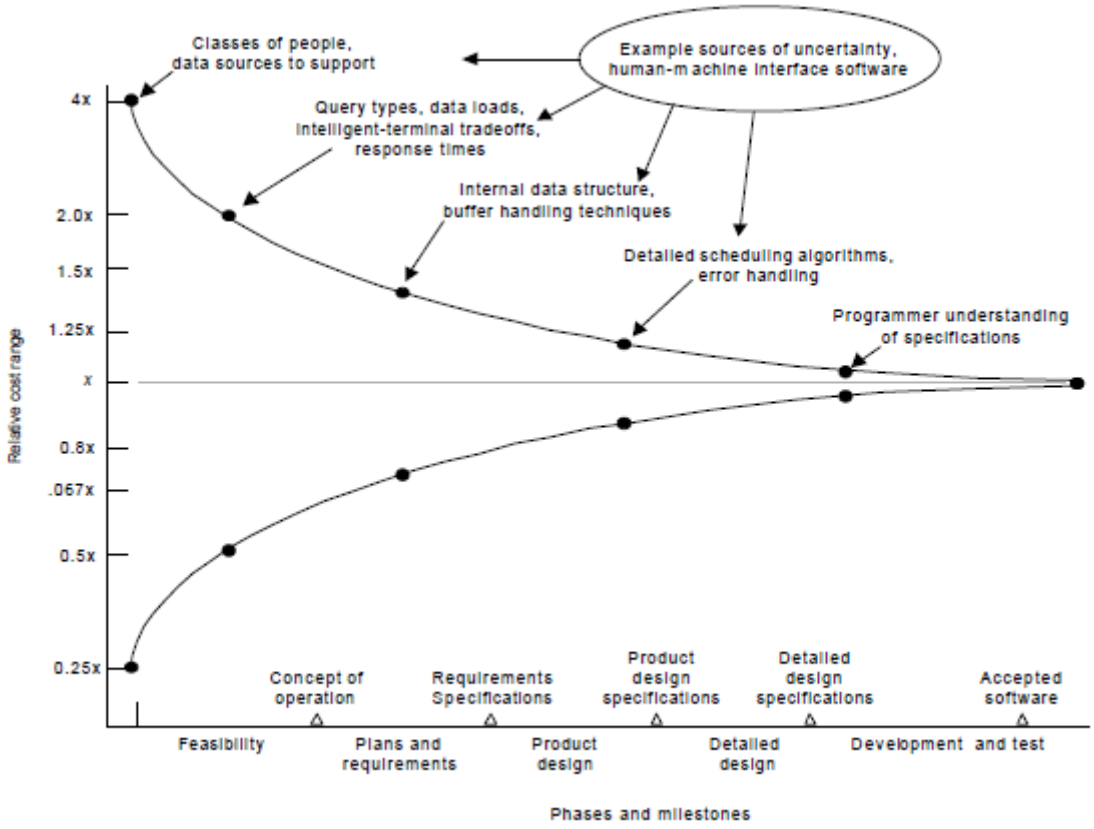

Figure 27 Software Effort Estimation Accuracy Versus Phase [47]

Laranjeria [31] suggests that increasing detail on functional specification of the system will result in a similar behavior in estimation values. Figure 28 depicts the change boundary of estimation errors with increasing details in the functional specification of the system by means of decomposition, during the course of the software development lifecycle.

Differing from Laranjeria'a approach, we suggest that only the bottom level for a decomposition can be defined absolutely and therefore pose a reference rather than the top level. It is possible to measure software size using the lowest level of decomposition for data movement, automatically. (Sequence diagrams, movement count). Then, it is possible to scale up the measurement by mapping lower decomposition level objects to objects in higher levels.

Based on the historical data on past projects, the characteristics of this decay in the error with respect to decomposition level can be determined. Then, organizations can

identify an optimum point for the level of decomposition needed for estimations based on their need of accuracy and the effort/time needed to decompose the system in the required level.

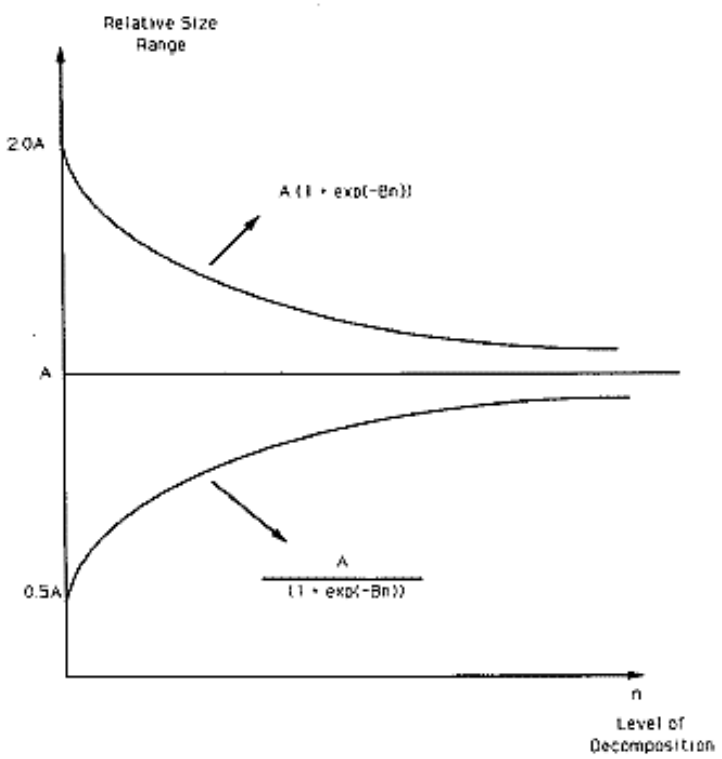

Figure 28. Software size estimation accuracy as a function of object decomposition level in the functional model. [31]

4.6.1. Structural Decomposition

FSM methods such as COSMIC address the level of granularity by the definition for Functional Process. However, this definition does not specify a granularity level for the behavior of the system. Instead, it defines how to select a consistent set of actions defined in the Requirement Specifications and group them to be measured separately.

This approach however, assumes that (or "should be assuming that") the requirements to be measured are defined at the same level of granularity. It does not define a formal method to use the same level of granularity in the requirements.

The level of granularity and hence, the number of Objects of Interest directly affect the measurement result.

Definition of a system:

It is hard to distinguish concepts belonging to different intermediate levels of decomposition from each other. However, lowest level of decomposition (minimum size of granule) can be identified for a software system, from the development point of view.

Lowest level of decomposition from a developments point of view would be the code that is actually developed.

The logical level of abstraction lying within the code may differ based on the technology used. However, the development effort and the size of code will be the same from the developer's point of view.

Based on this, the lowest level of decomposition for Data Movements in an OO system would be the calls (and methods) of an object. There would be no lower levels of data movement to be developed.

Level of granularity is a concept for objects. (eg: OOI)

Level of abstraction is a concept for behavior.

It is possible to measure software size using the lowest level of decomposition for data movement, automatically. (Sequence diagrams, movement count). Then, it is possible to scale up the measurement by mapping lower decomposition level objects to objects in higher levels

This way, an organization can effectively identify the (highest) level of decomposition that will suit their estimation needs. It will be the optimum level dictated by the time in the project lifecycle the organization needs the estimation and the level of expected estimation error.

CHAPTER 5

VALIDATION

Every line is the perfect length if you don't measure it.

— Marty Rubin

Baker et al. define the validation of software measurement in two dimensions as internal and external validation, and separates the validation of a prediction system from the validation of the measurement method [34]:

Internal Validation: Validation of a software measure is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute? This type of validation is central in our use of measurement theory. Practitioners may prefer to regard this as ensuring the well definedness and consistency of the measure? To stress this where necessary we may also refer to it as internal validation since it may require consideration of the underlying models used to capture the objects and attributes?

External Validation: External validation of a measure m is the process of establishing a consistent relationship between m and some available empirical data purporting to measure some useful attribute.

Validation of a Prediction System: Validation of a prediction system is the usual empirical process of establishing the accuracy of the prediction system in a given environment by empirical means, i.e., by comparing model performance with known data points in a given environment.

5.1. Validation Method

The verification activities can be listed according to the various design levels.

5.1.1. Modeling the empirical world

At this level, one should ensure that the characteristics formulated for the empirical world actually represent the concept to be modeled, and that this representation is correct. The verification issues are:

On the one hand, one should be sure that the model built actually corresponds to the consensual representation of the domain practitioners have about the attribute to be measured.

On the other hand, if a mathematical description is given through axioms, some internal properties of the model elaborated (e.g. completeness, consistency) should be verified, as in any mathematical description. If the description is given through a graphical modeling language, the model should also respect some internal rules. For example, an attribute model described through UML diagrams should respect rules like connectivity.

5.1.2. Modeling the numerical world

The selected mathematical structure should preserve the properties of the empirical world. Mathematically, this means that the numerical structure is homomorphic with the empirical structure, i.e. the mapping between the empirical structure and the mathematical structure is a homomorphism.

General ordering properties, like reflexivity, transitivity and antisymmetry, should be checked to validate whether appropriate scale types are used in the design of the measurement method [59] [75], [105].

5.1.3. Defining the measurement method

Verifying the mapping characteristics: According to the theoretical view, a method implements a homomorphic mapping between the empirical world and the numerical world.

In practice, a method to measure one attribute should produce results that correspond to the knowledge practitioners have about that attribute, that is, the ordering induced by that attribute (e.g. the "complexity" attribute and the ordering relation "A is more complex than B"). But, the characterization of the ordering through models is not sufficient, and verification through experimentation is thus necessary.

Verifying the assignment rules (the operations sequences that describe a measurement method) also involves other activities depending on the way those rules are expressed. In all cases, the procedural description (e.g. the counting rules, the operational mode of a particular measuring instrument, etc.) should be verified to ensure it embodies the measurement principle.

One approach to complete mathematical reasoning through experimentation is suggested by Melton et al. [81] and developed by Lopez et al. [79]. The idea is to complement the characterization of the attribute through a representative set of entities, which should be selected and ordered by domain experts. An experimental verification can thus be achieved on that finite set to check whether or not the ranking is indeed preserved by the measurement mapping.

5.2. Case Studies

In Chapter 1, we summarized the problems observed with current functional size measurement methods, our solution approach and research goals. Through DMP method, we intend to address problems of reliability, granularity, effort estimation and benchmarking.

In order to validate our solution to these problems and research goals we set in Chapter 1; we defined these goals for the case study we performed.

Case Study Goals:

1. Validate that it is possible to measure existing software products by backfiring measurable software models from source code.
2. Validate that DMP is a better input for effort estimation than problem domain sizes.
3. Investigate DMP's representation of project effort vs. solution domain sizes'.
4. Validate homomorphism of DMP by a reliable software measurement.
5. Validate that DMP method results are reliable and repeatable.
6. Validate that DMP is easier to learn and use than other FSM methods.

5.2.1. Case study design

Research Questions

In order to meet the validation goals, we defined research questions to be investigated through case studies.

Goal 2, 3:

1. What is the correlation of DMP results with project effort?
2. How does this correlation compare with effort correlation of problem and solution domain sizes?

Goal 4:

3. What is the correlation of DMP with LOC size?
4. How does DMP perform to predict LOC compared to other FSM methods?

Goal 5:

5. How reliable or repeatable are DMP results?

Goal 6:

6. How much effort does it take to measure software with DMP?
7. How is this effort compared to other common measurement methods?
8. How much effort does it take for software engineers to learn and apply the DMP method?

5.2.2. Case Study Plan

In order to investigate the research questions, we needed projects for which we could measure all problem domain size, solution domain size and DMP and have the project effort data.

For problem domain size, we chose to use COSMIC measurements. Reasons for choosing COSMIC was:

- We have experience with the method
- Easier to apply than other FSM methods
- It is commonly used.
- We have performed studies on its reliability
- There are studies investigating COSMIC's representation of effort data.

For solution domain size, we chose to use SLOC measurements. Reasons for choosing SLOC was:

- Easy to measure, as it is measured from source code through software tools
- Reliable, as it is counting concrete constructs instead of an abstract model, and repeatable
- Most of the common effort estimation methods utilize SLOC as input

- The measurement that correlates best with development effort

Then we planned to investigate the correlation of measurements mentioned above with the project effort.

We also planned to observe the measurement process and record the time it took to measure the projects.

Case Selection

In order to increase the number of samples and investigate the performance of DMP in different software development environments, we conducted four case studies in four different environments. We included student projects along with the projects in the industry as based on our experience, it is hard to access enough number of projects conducted in the industry that have a proper and complete requirements documents to conduct COSMIC measurements and/or complete design documents for the whole scope of a project.

First case study was conducted in a company which maintains a big MIS software framework. The company releases monthly releases. That is, the development timeframe and effort is fixed for each release. They include new features, bugs fixes and changes in existing modules.

The second case study was conducted with student projects. Different student groups were required to develop a software product with the same purpose and a common problem definition.

The third case study was conducted with an IT department of a governmental institution.

The fourth case study was conducted with a single very big simulation project developed for the defense industry.

D**ata Collection**

In order to conduct COSMIC, SLOC and DMP measurements for a project, we needed to have:

- Software Requirements for COSMIC measurement.
- Source code for SLOC measurement.
- Design models (component, class, sequence diagrams) and source code for DMP in various tiers.

We planned to measure COSMIC size from the Software Requirements Specification documents for the projects. Student projects already had this measurement done.

We planned to measure both physical and logical SLOC from the source code. We decided to use the same measurement tool for every project for consistency. We planned to use the tool "LOC Metrics".

We planned to measure for DMPs for each tier identifiable from the work products at hand. We would use the SDMC tool to count for DMPs from the source code of the projects. Than we would calculate higher tier measurements by grouping lowest level entities based on the design documents.

The approximation of DMPs for estimation can only be conducted during the course of a project by the software engineers designing and developing the software, it was not possible to investigate size approximations for DMP.

5.2.3. Case Study Execution

In this section, we summarize the conduct of each case study. We define the projects we investigated, define collection of data and the environment for the projects. Detailed measurement and data analysis results are given in Appendix A.

Case 1 – MIS Framework

Projects: We measured a projects which is developed separately and added to the framework based on a monthly timeframe. Project had a client-server architecture and both sides had their own releases. We investigated 4 releases for client and server software for the project 8 data sets.

We had access to source code and the end product. Requirements were not of consistent granularity and quality. There were no design documents.

COSMIC measurement: As requirements were not of consistent granularity and quality and was missing for common functionalities developed within the company, we could not get COSMIC measurements for the projects.

SLOC measurement: Physical and logical SLOC measurements were conducted using the Loc Metrics tool from the release baselines of the source codes. We measured the total size of the client and server side software as well as the size of each component.

DMP measurement: DMP measurements were conducted using the SDMC tool. As the projects involved many components and layers, modeling the source code by lowest level sequence diagrams took longer than expected.

The measurement was conducted at Tier Level 0 as the decomposition information for the projects were not defined as per the requirements of the DMP method. Instead we identified the major components from the source code based on the deployment, references, class diagrams and component diagrams which we reverse engineered from the source code. We measured the total project size and identified component's size in DMPs separately.

Effort Data: The effort value was constant for each release as there was a set time frame. The difference in the sizes of each release is considered as the developed software size as there were no changes or deletions during the development period.

Development environment: Software was developed with c# on .Net framework with a client-server architecture. Development teams consisted of 7 and 4 people respectively.

Case 2 – Student Projects

Projects: We had access to all the work products such as SRS documents, design documents and source code. We investigated 3 projects.

COSMIC measurement: COSMIC measurement was conducted based on the SRS documents supplied.

SLOC measurement: Physical and logical SLOC measurements were conducted using the Loc Metrics tool. We measured the total size of the software as well as the size of each development component.

DMP measurement: DMP measurements were conducted using the SDMC tool. The measurement at Tier Level 0 was conducted based on the source code. Higher tier measurements were conducted based on the design models (class diagrams and sequence diagrams) supplied by the students. We measured the total project size and identified component's size in DMPs separately.

Effort Data: Effort information was deducted from project plans supplied by the students.

Development environment: Projects had different programming languages and frameworks. Teams consisted of 4 students.

Case 3 – Governmental Organization IT Department

Projects: We investigated 5 independent projects of varying sizes. We only had access to source codes and effort information in the resolution of man months. The projects

had similar architectures and were developed using a Model View Controller (MVC) pattern.

COSMIC measurement: As we did not have access to requirements we could not measure COSMIC sizes.

SLOC measurement: Physical and logical SLOC measurements were conducted using the Loc Metrics tool. We measured the total size of the software as well as the size of each development component.

DMP measurement: DMP measurements were conducted using the SDMC tool. The measurement at Tier Level 0 was conducted based on the source code.

The measurement was conducted at Tier Level 0 as the decomposition information for the projects were not defined as per the requirements of the DMP method. Instead we identified the major components from the source code based on the deployment, references. We measured the total project size and identified component's size in DMPs separately.

Effort Data: We had the effort data for projects in the resolution of man months.

Development environment: 3 projects were developed in c# language and in .Net framework. 2 projects were developed in Java.

Case 4 – Single Big Defense Industry Project

Project: The project was about a simulation system developed for defense industry. We had access to requirements, design, source code and effort data.

COSMIC measurement: COSMIC measurement was conducted based on the SRS document.

SLOC measurement: Physical and logical SLOC measurements were conducted using the Loc Metrics tool. We measured the total size of the software as well as the size of each development component.

DMP measurement: DMP measurements were conducted using the SDMC tool. The measurement at Tier Level 0 was conducted based on the source code. Higher tier measurements were conducted based on the design models (class diagrams and sequence diagrams) supplied by the students. We measured the total project size and identified component's size in DMPs separately.

Effort Data: We had the effort data for the whole project.

Development environment: The project was developed in c# language and in .Net framework.

5.2.4. Case Study Results

Effort Estimation

Table 9 Project Effort vs. COSMIC LOC and DMP Sizes

Case	Project	Effort (MM)	COSMIC	LOC	DMP
Case 1	Client Release 1-2	4	-	719	9
	Client Release 2-3	4	-	2166	2277
	Client Release 3-4	4	-	646	353
	Server Release 1-2	3	-	23	0
	Server Release 2-3	3	-	735	275
	Server Release 3-4	3	-	342	249
Case 2	Project 1	12	346	294	3114
	Project 2	15	280	2262	8806
	Project 3	18	697	2886	23873
Case 3	Project 1	1	-	3260	1497
	Project 2	1	-	2739	2109
	Project 3	3.5	-	13134	9936
	Project 4	1.5	-	4926	5236
	Project 5	4	-	11103	14398
Case 4	Project 1	42.75	1999	61459	17807

Table 10 Comparison of DMP, COSMIC and SLOC Correlation with Project Effort

Case	COSMIC	DMP	LOC
Case 1	NA	0,444131	0,602961
Case 2	0,782975	0,957993	0,967648
Case 3	NA	0,977221	0,95916
Case 4	NA	NA	NA
Overall	NA	0,652766	-0,11273
Overall for Case 2&4	0,782975	0,967648	0,957993

As one can see from the Table 10, LOC size correlates better with effort than both COSMIC and DMP within an organization. However, when we look at the overall correlation, that is, cross organization effort correlation, DMP performs better than LOC. This result is expected as changes in development technology and environment impacts the LOC the most. DMP on the other hand, is less prone to this effect as it is more abstract in nature.

In each case DMP's correlation was higher than COSMIC's. Although the number of data points for COSMIC was low, this result also backs up our initial claims.

We believe low correlation values in case 1 due to the fact that each iteration is not a project itself and we assumed a fixed development effort for one month and assumed the difference in size of releases will be the size of software developed within that month.

Predicting Solution Domain Size (LOC)

Below, are scatter diagrams for LOC size versus DMP size for each case study. LOC size for each component within each project was compared with their DMP tier 0 sizes. Components were identified based on deployment information gathered from the code and design documents. Detailed data tables can be found in Appendix A.

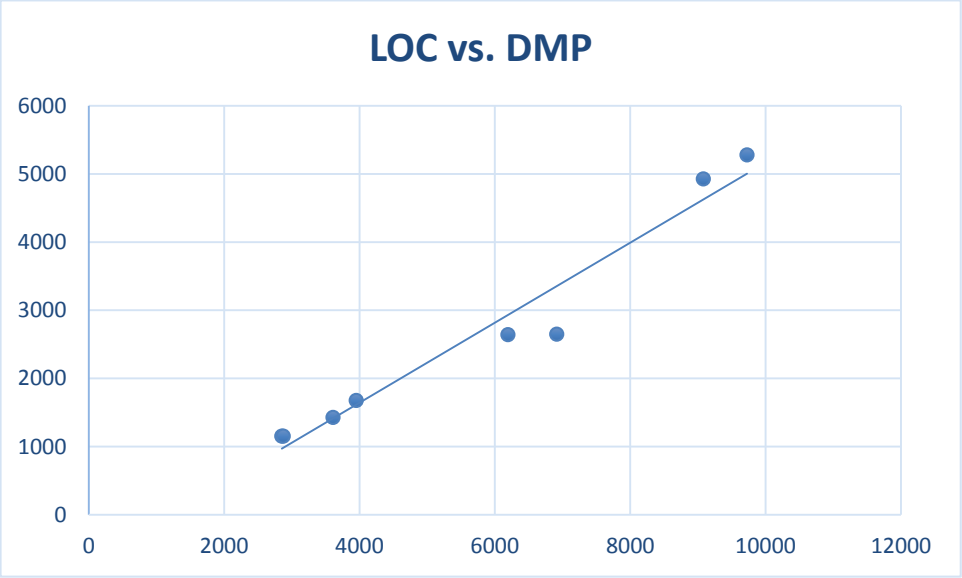

Figure 29 LOC vs. DMP Size - Case 1

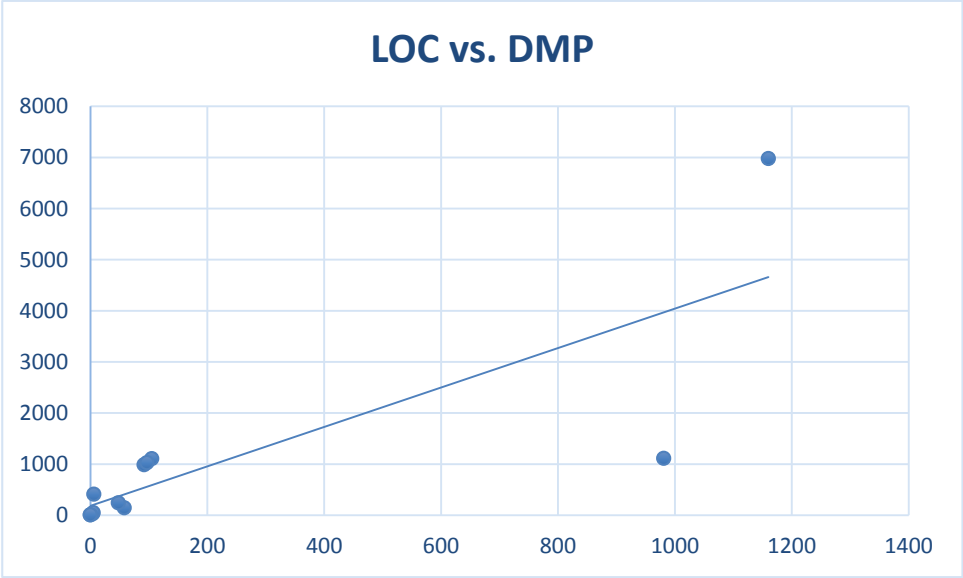

Figure 30 LOC vs. DMP Size - Case 2

Figure 31 LOC vs. DMP Size – Case 3

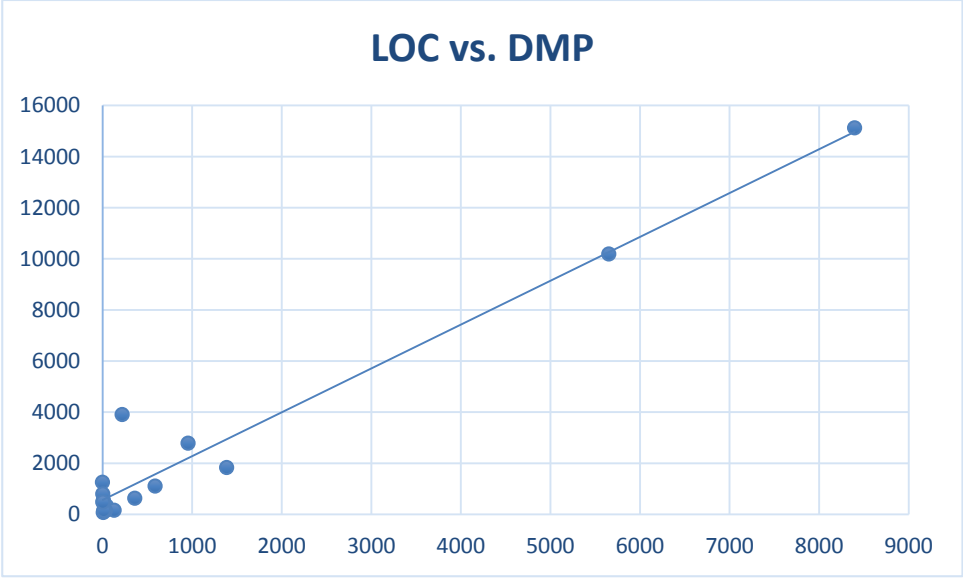

Figure 32 LOC vs. DMP Size - Case 4

Correlation of LOC size with DMP size for each case study is given in the table below. As one can see, DMP measurements do have a good correlation with LOC sizes. The correlation of COSMIC size with LOC however is found to be much lower both in our study and other studies in the literature. Based on this, we may say that DMP measurement is a better base for predicting LOC for the purpose of effort estimations.

Table 11 DMP -LOC Correlation Values

Case	DMP - LOC Correlation Value
Case 1	0,978373
Case 2	0,805721
Case 3	0,661689
Case 4	0,975973
Average	0,85544

Moreover, we may conclude that DMP measurement method, meets our initial aim of bridging the gap between solution domain and problem domain measurements.

Reliability and Repeatability of DMP Results

As far as tier 0 measurements are concerned, the DMP method was perfectly repeatable as it is performed on a concrete and exact input such as the source code.

For higher level tiers, however, effect of human error and individual interpretations were observed. Discrepancies between individual measurements in higher tiers were not caused by the counting process but by misidentification of the decomposition levels of structural entities.

These errors of mapping entities to tiers were actually self-healing. That is, if there is a common understanding of layers and components for a software project or framework within the project team or the whole organization, these errors are supposed to be ironed out and tier definitions will be used consistently.

Unfortunately, we could not collect enough data points for individual measurements to perform a quantitative analysis. We discussed the results and errors with measures and observe the abovementioned characteristics.

Based on our previous studies [101] and studies in literature [50][49], In tier 0, DMP is definitely much more reliable and repeatable then any other FSM method. The higher the tier the less reliable and repeatable the measurements get as expected.

Measurement Effort with DMP

We have observed that, modeling the source code from the sequence diagram for tier 0 measurements can take time as long as a couple of hours based on the size of the code, number of structural entities, layers and the computation power of the computer the SDMC tool is running on. However, as the measurement was performed automatically, the human effort needed to conduct the measurement was very small.

For higher Tier measurements, the measurement time and effort is less as the size of higher level models and hence their DMP size are lower. However, in the industry companies tend to omit detailed design phases or require design models only for complex elements of the project.

On the other hand, FSM measurement (e.g COSMIC, IFPUG) takes much more time and human effort. In our previous studies, we had calculated the average effort for COSMIC measurements as 2 minutes per FP for experts and much higher for inexperienced measurers [101]. DMP is a great improvement on other FSM methods for measurement effort.

SLOC measurement on the other hand, took time in the degree of seconds which was negligible.

Learning Curve for DMP

We had the chance to let software engineers perform the measurement in Cases 1 and 3. As far as Tier 0 measurements are concerned, it took a negligible time for them to learn how to use SDMC and measure DMP from the source code.

We had explained the rationale of the method and told them how to use it for higher tiers in previous phases of a project. Those with experience in software design and development had no difficulties in practicing the method. However, those with experience in only analysis had difficulties on predicting the tier level of a measurement performed in earlier stages.

Nonetheless, compared with our previous experiences in COSMIC training, measurers learned to DMP method in much less time and with higher success. In case 1 they mentioned it was *natural* for them to *predict how deep an entity will go* in decomposition as they are accustomed to *imagine further levels of development* while they are writing requirements or developing *pseudo designs* for the requirements.

5.2.5. Validity Threats

Different organizations have different definitions for project effort. The method of collecting the effort data also differs. Cross case evaluations based on effort data may have less accuracy compared to evaluations within a single organization.

In Case 2, student projects were investigated. Being class assignments, the quality of project documents and accuracy of project data may be lower than those collected from actual projects in the industry.

Higher tier DMP measurements are a part of project estimation process and identified with the prediction of further developments within a project. These measurements

can be performed through the course of a newly developed project and by those who are to design and develop the software. Therefore, it was not possible to perform such measurements within the case studies. However, higher tier DMP measurements are essentially logical subsets of tier 0 measurements and thus, any representation value of tier 0 measurements can also be attributed to higher tier measurements as well. In further studies, estimation methods based on DMP measurements will be investigated.

CHAPTER 6

CONCLUSION

There's no point in being exact about something if you don't even know what you're talking about.

--John von Neumann

In this thesis, we addressed reliability issues, granularity issues and benchmarking issues in functional size measurement methods. We also addressed the issues in effort estimation using functional size as an input.

We proposed a measurement framework to address these problems and meet the research goals. The framework consists of:

- A measurement method for sizing software systems from detailed design models.
- A method for sizing existing software products by backfiring measureable software models from software code.
- An approximation approach / representation for measurement results including the abstraction level the measurement is performed in.
- An estimation approach for size and effort.

The primary research goal were reached through the designed framework. Second goal of minimizing the measurement error and inter-measurer variance was attained through automating the measurement procedure and backfiring atomic level data movements from code. Third goal was also embedded in the measurement framework

The main contributions of this thesis is the incorporation of decomposition to the functional size measurement method and measurement results and basing the measurement on a concept that is common in both problem and solution domains.

6.1. Contributions

Streamlining the Measurement Concepts

We defined a measurement approach based on a concept that is common in both problem and solution domains; data movement. This streamlines measurement of both project and product attributes in two domains. This improves conversion of units, estimations, approximations and normalization of several size definitions and values.

Improvements in Reliability of Measurement Results

Subjective identification of abstract FSM concepts such as Objects of Interest and functional processes comprise the reliability of measurements. We define an atomic and objective definition of functional process, object and data movement. Through use of a measurement tool, we backfire these concepts of FSM from the source code which is the end point of solution implementation. This totally eliminates the reliability issues.

For higher level measurement results we traverse through higher levels utilizing metadata for the lowest level. Only point of human interpretation is in the generation of this metadata which relates lower level concepts to higher level ones. This mitigates measurement errors as there is less room left for interpretation and renders errors recoverable by fixing the metadata.

Resilient Measurement

Existing FSM methods follow a top down approach in modeling. Functional size measurement methods in the literature first develop an abstract model for a system definition for measurement purposes and then conduct measurement on that model. This abstraction needs to model the whole system correctly to make a successful measurement. Imperfect, partial or incomplete system definitions result in erroneous measurement models and this in turn have a big impact on the measurement results as the measurements use this model as a basis and use in every step of measurement procedure.

However, DMP method have a bottom up modeling approach. The measurement model is based on atomic level of decompositions. This makes the measurement results less susceptible to erroneous and incomplete system definitions. Missing parts of a system will not affect the other parts and aspects of the measurement model. Only error in the measurement results will be missing size for the missing definitions. This makes the DMP model much less susceptible to imperfect, partial or incomplete system definitions.

6.2. Significance of the Study

Better Effort Estimations

Most estimation models in the field dictate using several factors and multipliers to convert problem domain sizes to solution domain sizes and utilize historical data to estimate the project effort based on the solution domain size.

Assuming an inherent relation between different size measurements in different domains and predicting one using other actually introduces another level of estimation error. We suggest an estimation approach which rely on the same concepts that the measurement method does will eliminate the gap caused by such conversions and by this approach, estimations will become less prone to gaps between domains and project phases (see section 4.2.1).

Moreover, most FSM methods either does not include data manipulations in measurement or just incorporate the size of manipulations as an order of complexity to the overall measurement. As discussed in section 2.2 manipulations defined in a system gradually become movements as decomposition levels deepen. By measuring in lower levels of decomposition, DMP method measures data manipulations which would otherwise be left out in higher levels into measurement. This should also increase the accuracy of estimations.

Better Measurement of Software Changes

By DMP method, it is possible to measure specifications defined in levels lower than functional user requirements. This will make measuring software changes that are defined in lower resolution levels than functional user requirements more accurate than existing FSM methods.

Identifying the tier level of change requests will also improve the change management processes as the scope and impact of the change can be better analyzed.

Better Benchmarking

Having decomposition level incorporated into measurement results makes the scope and abstraction of the measured software product visible. This is especially crucial in benchmarking studies as current benchmarking datasets either do not include this information or do not have predefined scale for decomposition level.

Comparing measurement methods on the same level of decomposition will greatly improve the accuracy of benchmarking studies.

6.3. Future Work

We believe that it is possible to apply the measurement approach defined in the thesis to other system definition paradigms.

If we move up in the decomposition levels of system definition, the same approach can be applied to system specification level or business process definitions.

Another PhD thesis is being conducted in our research group on sizing business process models based on functionality. Business process models are problem domain models in higher levels of abstraction then system or software specifications. It is possible to extend the problem-solution flow to the level of business processes. This way, we believe it will be possible to have a measurement approach based on same concepts of functional process and data movement starting from business processes down to implementation of classes. This would be a great opportunity to build a streamlined measurement and estimation framework spanning the whole software engineering process.

REFERENCES

[1] M. Jackson, *Software requirements & specifications: a lexicon of practice, principles and prejudices*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1995.

[2] R. Dumke and A. Abran, *COSMIC Function Points: Theory and Advanced Practices*. 2011.

[3] M. A. Talib, A. Abran, and O. Ormandjieva, "MARKOV MODEL AND FUNCTIONAL SIZE WITH COSMIC-FFP," pp. 3240–3245, 2006.

[4] C. Gencel, "How to Use COSMIC Functional Size in Effort Estimation Models?," pp. 196–207, 2008.

[5] L. De Rore, M. Snoeck, G. Poels, G. Dedene, L. De Rore, M. Snoeck, and G. Poels, "Cocomo II as productivity measurement: a case study at KBC Cocomo II as productivity measurement: a case study at KBC."

[6] B. Ozkan, O. Turetken, and O. Demirors, "Software Functional Size: For Cost Estimation and More," pp. 59–69, 2008.

[7] "General Services Administration Acquisition Manual (GSAM) - Chapter 13: Software Estimation, Measurement, and Metrics," 2004.

[8] A. Abran, *Software metrics and software metrology*. John Wiley & Sons Interscience and IEEE-CS Press, New Jersey, 2010.

[9] R. Of, T. Draft, A. R. E. Invited, T. O. Submit, W. T. Comments, N. Of, A. N. Y. Relevant, P. Rights, O. F. Which, T. A. R. E. Aware, I. N. Addition, T. O. Their, E. As, B. Acceptable, F. O. R. Industrial, U. Purposes, D. Guides, M. A. Y. On, O. Have, T. O. Be, C. In, T. H. E. Light, O. F. Their, P. To, B. Documents, and T. O. Which, "International vocabulary of basic and general terms in metrology (VIM)," no. Dguide 99999, 2004.

[10] A. Abran, "Software Metrics and Software Metrology," May 2010.

[11] N. Habra, A. Abran, M. Lopez, and A. Sellami, "A framework for the design and verification of software measurement methods," *J. Syst. Softw.*, vol. 81, no. 5, pp. 633–648, May 2008.

[12] ISO, "ISO/IEC 14143-1- Information Technology - Software measurement - Functional Size Measurement. Part 1: Definition of Concept." 1998.

[13] A. Abran, "Software Metrics Need to Mature into Software Metrology (Recommendations) Position paper prepared by Software Metrics Need to Mature into Software Metrology (Recommendations)," pp. 1–18, 1998.

[14] L. Lavazza and V. Del Bianco, "A Case Study in COSMIC Functional Size Measurement: the Rice Cooker Revisited."

[15] C. Symons, "COSMIC GROUP CASE STUDY: RICE COOKER," pp. 1–15, 2010.

[16] C. Gencel, R. Heldal, and K. Lind, "On the Relationship between Different Size Measures in the Software Life Cycle Cigdem," 2009.

[17] M. Staples, R. Kolanski, G. Klein, C. Lewis, J. Andronick, T. Murray, R. Jeffery, and L. Bass, "Formal specifications better than function points for code sizing," *2013 35th Int. Conf. Softw. Eng.*, pp. 1257–1260, May 2013.

[18] H. Diab and M. Frappier, "Formalizing COSMIC-FFP Using ROOM .," no. Cmm, 2001.

[19] F. K. R. S. H. Diab, "µcROSE: Functional Size Measurement for Rational Rose RealTime Topic: Embedding metrics in OO CASE tools."

[20] G. Poels, "Definition and Validation of a COSMIC-FFP Functional Size Measure for Object-Oriented Systems," pp. 1–6, 2003.

[21] M. S. Jenner, "Automation of Counting of Functional Size Using COSMIC FFP in UML Functional users requirements Use case Functional process type Sub-process," pp. 43–51.

[22] N. Condori-Fernández, S. Abrahão, and O. Pastor, "On the Estimation of the Functional Size of Software from Requirements Specifications," *J. Comput. Sci. Technol.*, vol. 22, no. 3, pp. 358–370, May 2007.

[23] E. G. T. S. K. S. Piotr Habela, "Serafinski T., Adapting Use Case Model for COSMIC-FFP based Measurement."

[24] G. Levesque, V. Bevo, and D. T. Cao, "Estimating software size with UML models," in *Proceedings of the 2008 C3S2E conference on - C3S2E '08*, 2008, p. 81.

[25] O. P. G. G. Beatriz Marín, "Automating the Measurement of Functional Size . . ."

[26] G. Li and S. Yao, "Research on Mapping Algorithm of UML Sequence Diagrams to Object Petri Nets," pp. 285–289, 2009.

[27] G. Lévesque and V. Bevo, "Measuring Size for the Development of A Cost Model : A Comparison of Results Based on COSMIC FFP and SLIM Back-Firing Function Points," no. Figure 2, pp. 197–205, 2001.

[28] M. S. Jenner, "Backfiring COSMIC size from Java and C ++ code."

[29] V. Del Bianco, L. Lavazza, and C. Politecnico, "An Empirical Assessment of Function Point-Like Object-Oriented Metrics," no. Metrics, 2005.

[30] V. Bianco and L. Lavazza, "Applying the COSMIC Functional Size Measurement Method to Problem Frames," pp. 282–290, 2009.

[31] L. A. Laranjeira, "Software size estimation of object-oriented systems," *Softw. Eng. IEEE Trans.*, vol. 16, no. 5, pp. 510–522, 1990.

[32] O. Neill, C. Symons, and M. O. Neill, "Software Functional Size with ISO 19761 : 2003 COSMIC-FFP Measurement Method Case Study : ******** Valve Control System ********** January 2006," no. January, 2006.

[33] O. D. Yalın Meriç, Erdir Ungan, "Automated Functional Size Measurement Using Sequence Diagrams," METU, 2013.

[34] A. L. Baker, J. M. Bieman, F. Collins, N. Fenton, D. A. Gustafson, A. Melton, and R. Whitty, "A Philosophy for Software Measurement," pp. 1–9.

[35] L. Buglione, "COSMIC software functional size," in *Automative SPIN*, 2010.

[36] A. Abran and J. P. Jacquet, "A Structured Analysis of the new ISO Standard on Functional Size Measurement - Definition of Concepts", 4th IEEE Int. Symposium and Forum on Software Engineering Standards, 1999

[37] A. Abran and P. N. Robillard, "Function Points: A Study of Their Measurement Processes and Scale Transformations", Journal of Systems and Software, vol. 25, pp. 171-184, 1994.

[38] A. J. Albrecht and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: a software science validation", IEEE Transactions on Software Engineering, vol. 9, no. 6, pp. 639–648, 1983

[39] A.J. Albrecht, and J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Transactions on Software Engineering, vol. SE-9, no. 6, November 1983.

[40] ASMA, "Sizing in Object-Oriented Environments", Australian Software Metrics Association (ASMA), Victoria, Australia 1994.

[41] Azzouz, S., Abran, A.: A proposed measurement role in the Rational Unified Process (RUP) and its implementation with ISO 19761: COSMIC FFP. In: Software Measurement European Forum 2004, Rome (2004)

[42] B. Boehm, Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[43] B. Kitchenham, "Using Function Points for Software Cost Estimation - Some Empirical Results", 10th Annual Conference of Software Metrics and Quality Assurance in Industry, Amsterdam, The Netherlands, 1993

[44] B. W. Boehm, Estimating Software Costs, Prentice Hall, N. J., 1981

[45] Bévo, V., Lévesque, G., Abran, A.: Application de la méthode FFP à partir d'une spécification selon la notation UML: compte rendu des premiers essais d'application et questions. In: 9th International Workshop Software Measurement, Lac Supérieur, Canada, pp. 230–242 (1999)

[46] Bevo, V.: Analyse et Formalisation Ontologique des Procédures de Mesure Associées aux Méthodes de Mesure de la Taille Fonctionnelle des Logiciels: de Nouvelles Perspectives Pour la Mesure. Doctoral thesis, Université du Québec à Montréal - UQAM, Montréal (2005)

[47] Boehm, Barry W., Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981

[48] C. Dekkers, and I. Gunter, "Using Backfiring to Accurately Size Software: More Wishful Thinking Than Science?", IT Metrics Strategies, Vol. VI, No.11, 2000, 1-8

[49] C. F. Kemerer, "An Empirical Validation of Software Cost Estimation Models", Communications of the ACM, vol. 30, no. 5, pp. 416-429, 1987

[50] Condori-Fernández, N., Pastor, O.: An Empirical Study on the Likelihood of Adoption in Practice of a Size Measurement Procedure for Requirements Specification. In: 6th International Conference on Quality Software – QSIC, Beijing, pp. 133–140 (2006)

[51] Condori-Fernández, N., Pastor, O.: Evaluating the Productivity and Reproducibility of a Measurement Procedure. In: ER Workshops, pp. 352–361 (2006)

[52] Condori-Fernández, N.: Un procedimiento de medición de tamaño funcional a partir de especificaciones de requisitos. Doctoral thesis, Universidad Politécnica de Valencia, Valencia (2007)

[53] COSMIC Group: Rice Cooker – Cosmic Group Case Study. École de technologie supérieure, Université du Québec à Montréal - UQAM, Montréal (2003)

[54] D. Garmus and D. Herron, Function Point Analysis: Measurement Practices for Sucessful Software Projects, 2000

[55] D. J. Ram and S. V. G. K. Raju, "Object Oriented Design Function Points", First Asia-Pacific Conference on Quality Software, 2000.

[56] D. J. Reifer, "Let the Numbers Do the Talking", CrossTalk: The Journal of Defense Software Engineering, March 2002

[57] E. Rains, "Function Points in an ADA Object-Oriented Design", OOPS Messenger, vol. 2, no. 4, pp. 23-25, 1991.

[58] F. Minkiewicz, "Measuring Object Oriented Software with Predictive Object Points", Project Control for Software Quality, A. C. Rob Kusters, Fred Heemstra and Erik van Veenendaal, Ed.: Shaker Publishing, 1999.

[59] Fenton, N., Pfleeger, S.L., 1997. Software Metrics – A rigorous and Practical Approach, second ed. International Thomson Computer Press, London

[60] Ferens, Daniel V. and David S. Christensen, "Does Calibration Improve the Predictive Accuracy of Software Cost Models?", CrossTalk, April 2000

[61] G. Antoniol and F. Calzolari, "Adapting Function Points to Object Oriented Information Systems", 10th Conference on Advanced Information Systems Engineering (CAiSE'98), 1998.

[62] G. C. Low and D. R. Jeffery, "Function Points in the estimation and evaluation of the software process", IEEE Transactions on Software Engineering, vol. 16, no. 1, pp. 64-71, 1990

[63] G.S. Henderson, "The Application of Function Points to Predict Source Lines of Code for Software Development", An MSc Thesis submitted to Air Force Inst. of Tech., Wright-Patterson AFB, OH, Report Number: AD-A258447, AFIT/GCA/LSY/92S-4, 1992

[64] H. M. Sneed, "Estimating the Development Costs of Object-Oriented Software", 7th European Software Control and Metrics Conference, Wilmslow, UK, 1996.

[65] H. Zhao and T. Stockman, "Software Sizing for OO software development - Object Function Point Analysis", GSE Conference, Berlin, Germany, 1995.

[66] IFPUG, "Function Point Counting Practices: Case Study 3 - Object-Oriented Analysis, Object Oriented Design (Draft)", 1995

[67] Integration Definition for Function Modeling (IDEF0). Federal Information Processing Standards Publications (FIPS PUBS) Publication 183. 1993 December 21

[68] ISBSG Data Collection Questionnaire, http://www.isbsg.org/isbsgnew.nsf/webpages/-GBLData%20Collection%20Questionaires?opendocument

[69] ISO, "ISO/IEC 14143-1- Information Technology - Software measurement - Functional Size Measurement. Part 1: Definition of Concepts", 1998

[70] J. Kammelar, "A Sizing Approach for OO-environments", 4th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, Cannes, 2000.

[71] J. P. Jacquet and A. Abran, "From Software Metrics to Software Measurement Methods: A Process Model", 3rd Int. Standard Symposium and Forum on Software Engineering Standards (ISESS'97), Walnut Creek, USA, 1997

[72] Jenner, M.S.: Automation of Counting of Functional Size Using COSMIC-FFP in UML. In: 12th International Workshop Software Measurement, pp. 43–51 (2002

[73] Jenner, M.S.: COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML – Problems of Granularity. In: 4th European Conference on Software Measurement and ICT Control, Heidelberg, pp. 173–184 (2001)

[74] K. Kavoussanakis, Terry Sloan,UKHEC Report on Software Estimation,The University of Edinburgh December 2001

[75] Kitchenham, B., Pfleeger, S.L., Fenton, N., 1995. Towards a framework for software measurement validation. IEEE Transactions on Software Engineering 21 (12), 929–944

[76] L. Laranjeira, "Software Size Estimation of Object-Oriented Systems", IEEE Transactions on Software Engineering, vol. 16, no. 5, pp. 510-522, 1990.

[77] L. Molinié and A. Abran, "Software Outsourcing Contracts: An Economic Analysis based on Agency Theory", International Workshop on Software Measurement (IWSM'98), Québec, Canada, 1999

[78] Lokan C., Wright T, Hill P.R., Stringer M. Organizational Benchmarking Using the ISBSG Data Repository. IEEE Software September/October 2001

[79] Lopez, M., Paulus, V., Habra, N., 2003. Integrated Validation Process of Software Measure. International Workshop on Software Measurement, IWSM2003, Montreal, Shaker Verlag, September

[80] Mellor, S., Balcer, J.: Executable UML: A Foundation for Model-Driven Architecture. Addison Wesley, Reading (2002)

[81] Melton, A.C., Gustafson, D.M., Bieman, J.M., Baker, A.L., 1990. A mathematical perspective for software measures research. Software Engineering Journal 5 (5), 246–254.

[82] N. Thomson, R. Johnson, R. MacLeod, G. Miller, and T. Hansen, "Project Estimation Using an Adaptation of Function Points and Use Cases for OO Projects", OOPSLA'94 Workshop on Pragmatic Theoretical Directions in Object-Oriented Software Metrics, 1994.

[83] Nagano, S., Ajisaka, T.: Functional metrics using COSMIC-FFP for object-oriented real- time systems. In: 13th International Workshop on Software Measurement, Montreal (2003)

[84] O. A. Lehne, "Experience Report: Function Points Counting of Object Oriented Analysis and Design based on the OOram method", Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'97), 1997.

[85] Object Management Group. (2011, August) Object Management Group. [Online]. http://www.omg.org/spec/UML/2.4.1/

[86] Özden Özcan Top, Baris Özkan, Mina Nabi, Onur Demirörs: Internal and External Software Benchmark Repository Utilization for Effort Estimation. IWSM/Mensura 2011: 302-307

[87] P. G. Rule, "The Importance of the Size Software Requirements", Software Measurement Services Ltd. UK, 18 p. 2001.

[88] Pastor, O., Gómez, J., Insfrán, E., Pelechano, V.: The OO-Method Approach for Information Systems Modelling: From Object-Oriented Conceptual Modeling to Automated Programming. Information Systems 26, 507–534 (2001)

[89] Poels, G.: A Functional Size Measurement Method for Event-Based Object-Oriented Enterprise Models. In: 4th International Conference on Enterprise Information Systems – ICEIS, Ciudad Real, pp. 667–675 (2002)

[90] R. D. Banker, R. J. Kauffman, and R. Kumar, "An Empirical Test of Object-based Output Measurement Metrics in a Computer Aided Software Engineering (CASE) Environment", Journal of Management Information Systems, vol. 8, no. 3, pp. 127-150, Winter 1991-92.

[91] Rubin, H.: Worldwide benchmark project report, Rubin Systems Inc. 1995

[92] Standish_Group, "The 2003 CHAOS Chronicles", The Standish Group International, Inc. 2003

[93] T. C. Jones, Estimating Software Costs, McGraw-Hill, New York, 1998

[94] T. de Marco, Structured Analysis and System Spec$cation. Engle- wood Cliffs, NJ: Prentice-Hall, 1979.

[95] T. Fetcke, A. Abran, and R. Dumke, "A Generalized Representation for Selected Functional Size Measurement Methods", 11th International Workshop on Software Measurement, Montréal, Canada, 2001.

[96] T. Fetcke, A. Abran, and R. Dumke, "A Generalized Representation for Selected Functional Size Measurement Methods", 11th International Workshop on Software Measurement, Montréal, Canada, 2001

[97] T. Fetcke, A. Abran, and T. H. Nguyen, "Function point analysis for the OO-Jacobson method: a mapping approach", FESMA'98, Antwerp, Belgium, 1998.

[98] T. Rollo, "Functional Size measurement and COCOMO – A synergistic Approach". In Proc. of Software Measurement European Forum 2006, 2006, 259-267

[99] T. Uemura, O. Kusumoto, and I. Katsuro, "Function-point analysis using design specifications based on the Unified Modelling Language", Journal of Software Maintenance and Evolution: Research and Practice, John Wiley & Sons, Ltd., vol. 13, no. 4, pp. 223-243, 2001.

[100] UKSMA, "MK II Function Point Analysis Counting Practices Manual, Version 1.3.1", United Kingdom Software Metrics Association September 1998

[101] Ungan,E., Demirörs O., Top O.O, Özkan B. An Experimental Study on the Reliability of COSMIC Measurement Results.. Lecture Notes in Computer Science, Volume 5891/2009, 321-336.Springer Berlin / Heidelberg. 2009.

[102] Ünal, İ, Ungan, E, Demirörs, O. "The Effect of Implementation Technology on Software Development Effort: An Industrial Case". EPIC workshop, In the proceedings of International Symposium on Empirical Software Engineering and Measurement. Bolzano, Italy. 2010.

[103] Ünal, İ.: Predicting Effort from COSMIC FSM method or Design Size : A Case Study.Technical Report, Middle East Technical University, Ankara, 2010.

[104] Zoltán Micskei and Hélène Waeselynck, "UML 2.0 Sequence Diagrams' Semantics," Budapest University of Technology and Economics and Université de Toulouse, Budapest and Toulouse, Technical IST 026764, 2008.

[105] Zuse, H. (Ed.), 1998. Framework of Software Measurement. Walter de Gruyter

APPENDICES

APPENDIX A CASE STUDY RESULTS

Case 1 – MIS Framework

Table 12 LOC and DMP Sizes for Client Side Software Releases

	LOC	DMP
Project 1 Client Release 1	6195	2639
Project 1 Client Release 2	6914	2648
Project 1 Client Release 3	9080	4925
Project 1 Client Release 3	9726	5278

Table 13 LOC and DMP Sizes for Server Side Software Releases

	LOC	DMP
Project 1 Server Release 1	2852	1151
Project 1 Server Release 2	2875	1151
Project 1 Server Release 3	3610	1426
Project 1 Server Release 4	3952	1675

Case 2 – Student Projects

Table 14 LOC and DMP Sizes for Student Project Components

Project	Component	LOC	DMP
Student Project 1		3114	
	Component 1	986	92
	Component 2	1104	105
	Component 3	3	0
	Component 4	1024	97
	Component 5	3	0
Student Project 2		8806	2262
	Component 1	143	58
	Component 1.1	25	4
	Component 2	409	6
	Component 3	6976	1160
	Component 3.1	238	48
	Component 4	48	5
	Component 5	1110	981
Student Project 3		23873	
	Component 1	2566	
	Component 2	21307	

Table 15 Project Effort vs. COSMIC, DMP and LOC Measurements

Project	COSMIC	DMP	LOC	Effort (man.month)
Project 1	346	294	3114	12
Project 2	280	2262	8806	15
Project 3	697	3387	23873	18

Case 3 – Governmental Organization IT Department

Table 16 LOC and DMP Measurements for Project 1 Components

	LOC		DMP
Project 1	3260		1497
	Component 1	153	
	Component 2	1050	
	Component 3	288	
	Component 4	35	
	Component 5	634	
	Component 6	20	4
	Component 7	652	
	Component 8	428	

Table 17 LOC and DMP Measurements for Project 2 Components

	LOC		DMP
Project 2	2739		2109
	Component 1	80	127
	Component 2	207	294
	Component 3	86	127
	Component 4	34	48
	Component 5	20	4
	Component 6	28	37
	Component 8	40	88
	Component 9	260	265
	Component 10	388	1119

Table 18 LOC and DMP Measurements for Project 3 Components

	LOC			DMP
Project 3	13134			9936
	Component 1	4199		
		Component 1.1	1535	
		Component 1.2	1014	
		Component 1.3	48	
		Component 1.4	1597	
		Component 1.5	5	
	Component 2	8935		
		Component 2.1	485	
		Component 2.2	2532	
		Component 2.3	3342	
		Component 2.4	44	

Table 19 LOC and DMP Measurements for Project 4 Components

	LOC			DMP
Project 4	4926			5236
	Component 1	711		2079
	Component 2	49		0
	Component 3	767		1740
	Component 4	299		308
	Component 5	43		48
	Component 6	1853		
		Component 6.1	353	14
		Component 6.1	1500	18
	Component 9	311		166
	Component 10	893		863

Table 20 LOC and DMP Measurements for Project 5 Components

	LOC			DMP
Project 5	11103			14398
	Component 1	653		102
	Component 2	177		0
	Component 3	1534		3863
	Component 4	637		1057
	Component 5	34		65
	Component 6	76		305
	Component 7	9		0
	Component 8	6019		1948
		Component 8.1	4653	1325
		Component 8.2	178	76
		Component 8.3	571	128
		Component 8.4	617	419
	Component 9	85		34
	Component 10	931		4169
	Component 11	0		0
	Component 12	139		81
	Component 13	392		1402
	Component 14	417		1372

Case 4 – Single Big Defense Industry Project

COSMIC Size: 1999 CFP

Table 21 LOC and DMP Measurements for Project Components

Component Level 1	SLOC	DMP
Component 1	10184	5653
Component 2	3906	220
Component 3	340	43
Component 4	9992	
Component 5	631	360
Component 6	2488	
Component 7	7013	
Component 8	15125	8397
Component 9	1102	587
Component 10	157	130
Component 11	69	8
Component 12	2372	
Component 13	1829	1387
Component 14	2783	955
Component 15	147	34
Component 16	152	14
Component 17	1255	0
Component 18	552	11
Component 19	798	4
Component 20	478	4
Component 21	86	

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Ungan, Erdir

Nationality: Turkish (TC)

Date and Place of Birth: 1 April 1980, Ankara

Marital Status: Single

Phone: +90 505 6444596

Fax: +90 312 592 1043

email: erdir.ungan@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
MS	METU Computer Engineering	2006
MS	METU Industrial Engineering	2006
BS	METU Electrical and Electronics Eng.	2003
High School	Ari Fen Lisesi, Ankara	1996

WORK EXPERIENCE

Year	Place	Enrollment
2008-present	Bilgi Grubu Ltd.	Consultant, Trainer, Project Manager
2008-2011	METU Informatics Institute	Research Project Assistant
2006-2008	TAI (Turkish Aerospace Industries) Avionic Software Department	Software Process Engineer Erciyes Project (Avionic Modernization of C-130 planes): DO-178B Certification Verification Engineer
2005	TÜBİTAK (The Scientific and Technological Research Council of Turkey) : EU 6.Framework Programme National Coordination Office TR-MONET (Turkish Mobility Network) Project	Project Personel – Technical Expert
2004	PFAFF – Quick Rotan GmBH Kaiserslautern / Germany Research and Development Department	Software Developer
2002	DG Industrial Solutions Ltd. Home Technologies Department	Home Technologies Specialist Home Technologies Specialist Trainer
2001	Meteksan Computer systems Co.	Internship
1999	Ericcsson Turkey	Internship

FOREIGN LANGUAGES

Advanced English, Basic German

PUBLICATIONS

1. A Clustering Based Functional Similarity Measurement Approach.Usgurlu, B., Ozcan Top, O., Ungan, E. & Demirors, O. 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA). Lillie, France. 2010.
2. An Experimental Study on the Reliability of COSMIC Measurement Results. Ungan,E., Demirörs O., Top O.O, Özkan B.. Lecture Notes in Computer Science, Volume 5891/2009, 321-336.Springer Berlin / Heidelberg. 2009.
3. Common Practices and Problems in Effort Data Collection in the Software Industry. E. Ungan, A. Özkaya, O. Demirörs. IWSM / Mensura 2011. Nara, Japan. 2011.
4. COSMIC İşlevsel Büyüklük Ölçüm Sonuçlarında Gözlenen Sapmalar Üzerine Bir Deney Çalışması. Ungan, E , Ozkan, B., & Demirors, O. Ulusal Yazılım Mühendisliği Sempozyumu. Ankara 2009
5. Evaluation of Reliability Improvements for COSMIC Size Measurement Results Ungan, E., Ozcan Top, O., Ozkan, B. & Demirors, O . International Confrence on Software Measurement IWSM/MENSURA/MetriKon. Stuttgart, Germany. 2010.
6. Evaluation of Reliability Improvements for COSMIC Size Measurement Results Ungan, E., Ozcan Top, O., Ozkan, B. & Demirors, O . International Confrence on Software Measurement IWSM/MENSURA/MetriKon. Stuttgart, Germany. 2010.
7. İşlevsel Büyüklük Ölçüm Yöntemleri ve Referans Veri Kümelerinin Yazılım Mimarileri Açısından Değerlendirilmesi. O., Ozkan, B., Ungan, E & Demirors, O. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu 2012

8. Süreç İyileştirme Modellerinin Kamu Kurumlarında Uygulanabilirliği: Bir Durum Çalışması. Ozcan Top, O., Ungan, E., Cebeci, O., Demirörs, O. Ulusal Yazılım Mühendisliği Sempozyumu. Ankara 2012
9. The Effect of Implementation Technology on Software Development Effort: An Industrial Case.Unal, I., Ungan, E. & Demirors, O. International Symposium on Empirical Software Engineering and Measurement. Bolzano, Italy. 2010.
10. The Effect of the Quality of Software Requirements Document on the Functional Size Measurement.. G. Yılmaz, E. Ungan, O. Demirörs. United Kingdom Software Metrics Association International Conference on Software Metrics and Estimating. London, UK. 2011.
11. Yazılım Gereksinim Dokümanı Kalitesinin İşlevsel Büyüklük Ölçümüne Etkisi. G. Yılmaz, E. Ungan, O. Demirörs. Ulusal Yazılım Mühendisliği Sempozyumu. Ankara 2011.
12. Yazılım Projelerinde, Geliştirme Teknolojilerinin İşgücüne Etkisi. Unal, I., Ungan, E. & Demirors, O. Yazılım Kalitesi ve Yazılım Geliştirme Araçları Sempozyumu 2010.
13. Yazılım Sektöründe Efor Verisi Toplamanın Zorlukları ve Yaygınlığı. Ayşegül Özkaya, Erdir Ungan ve Onur Demirörs. V. Ulusal Yazılım Mühendisliği Sempozyumu. Ankara 2011.

Supervised Masters Projects

1. İlkay ünal - Predicting Effort From Cosmic FSM Method or Design Size: A Case - Technical report - METU/ii-tr-2010 - June 2010
2. Yalın Meriç - Automated Functional Size Measurement Using Sequence Diagrams – Technical report – Spring 2013

GRANTS/AWARDS

- Unified Software Estimation Method and Toolset – Research Project supported by The Scientific and Technological Research Council of Turkey(TUBITAK).

- Software Benchmark Dataset for Estimation and a Process Oriented Estimation Method supported by The Scientific and Technological Research Council of Turkey(TUBITAK).
- Success Scholarship for 3 years in METU Electrical and Electronics Eng. For being in the first 100 in the university entrance exam.
- 2nd place in Global Game Jam 2012 Turkey

HOBBIES

Playing the piano, Playing with a blues band, Board Games, Game Design, Medieval and WWII history, Science Fiction-Fantasy, Role-playing games, Paintball, Traveling.