UNIVERSITY OF GOTHENBURG

Predictive Software Measures Based on Formal Z Specifications

Master of Science Thesis in
Software Engineering and Management

Abdollah Tabareh

***Supervisors*:**

Dr. Miroslaw Staron

IT-universitetet i Göteborg
CHALMERS | GÖTEBORGS UNIVERSITET

Dr. Andreas Bollin

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT

University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, September 2011

Predictive Software Measures Based on Formal Z Specifications
Master of Science Thesis in Software Engineering and Management

Examiner: Associate Professor, Sven Arne Andreasson

University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden September 2011

Table of Contents

Abstract 1
Acknowledgment 2
Chapter 1: Introduction 3
1.1 Context 3
1.2 Scope 4
1.3 Value 5
1.4 Method 5
Chapter 2: Measures in Z Specifications 7
2.1 Objectives 7
2.2 Method 7
2.3 Formal Specification Measures 7
Chapter 3: Predictive Models 13
3.1 Introduction 13
3.2 Cost Estimation Approaches 13
3.3 Estimation Models 14
3.3.1 Input Parameters 15
3.4 Summary 16
Chapter 4: The Experiment 17
4.1 Introduction 17
4.2 Methodology 17
4.2.1 Subjects 17
4.2.2 Variables 19
4.2.3 Hypotheses 20
4.3 Results 20
4.4 Discussion 21
4.5 Threats to Validity 28
Chapter 5: Conclusions 29
5.1 Introduction 29
5.2 Study on Z Metrics 29
5.3 Study on Code Metrics 29
5.4 The Experiment 29
5.5 Further Studies 30
Appendix A- List of Study Subjects 31
Appendix B- Measurement Results 35
Appendix C- Analysis Results 40
References 44

Abstract

BACKGROUND: The success of software development projects depends highly on meeting the assigned schedule and budget of the project which are often defined in terms of a project plan. Estimation is the basis for planning; therefore, having a reliable way of estimating effort needed to perform the tasks is a must for a reliable project plan.

Already in 1987, Samson, Nevill and Dugard, showed that there is a strong and direct influence of formal specification metrics onto the effort needed for implementation. Since then, there has been some progress in various aspects of formal specifications; the introduction of specification slicing methods, slice-based specification metrics, and methods for visualization of specifications has opened new ways for measuring properties of specifications with more metrics. Nevertheless, there hasn't been much progress in the field of cost estimation using recent achievements of formal specifications.

METHODS: The main focus in this thesis work is to examine *if there is a correlation between formal Z specification measures and implementation related measures.* In concise, this work tries to explain the correlation between the measures in specifications and the measures in code which can be used as input parameters in currently existing software cost estimation models to estimate the total cost of software. This is examined through an experiment which is conducted via measuring 28 subjects using 11 metrics in specifications and 4 metrics in code.

CONCLUSION: The results of this thesis work show the size of code, which is the main input parameter of outstanding software cost estimation models, is predictable from formal Z specifications. There are proofs which show that 3 out of 4 investigated metrics in code are in correlation with the metrics in formal Z specifications.

Acknowledgment

First of all and foremost, I would like to thank my supervisor Dr. Andreas Bollin in Alpen-Adria Universität of Klagenfurt. His vision on this topic and his helps and guidance illuminated the way throughout performing this study. His concise explanations and examples were unweaving in problems of this work. In addition to technical assistance, his patience, sense of duty, and respectful manner impressed me.

I would like to thank Dr. Miroslaw Staron in Göteborgs Universitet. In spite of his heavy management responsibilities, he responded me in a timely manner and his precise reviews protected this work from many faults.

Lastly, I would like to thank my family, and my father in particular, whose supports and love was the main motivation for me to stand all the demoralizing problems during my master studies.

Abdollah Tabareh
Göteborgs Universitet
September 2011

Chapter 1:
Introduction

1.1 Context

In consequence of the increase in complexity of software systems, producing correct, reliable software has become a concern for software industry [1, p.56]. Software quality becomes a paramount aspect when it comes to safety-critical systems where human lives might be in danger. For example a defect in the navigation system of an airplane full of passengers or in a control system of a nuclear powerhouse can lead to a catastrophe. Despite of all concerns, we don't see these catastrophes too frequently. The fewer defects in these software systems are because of applying more precise methods throughout the development life cycle of these safety-critical software systems comparing to methods used for developing commercial software like iPhone applications.

Formal methods are rigorous techniques based on mathematical notation that can be used to specify and verify software models [3, p.268]. Formal methods provide a rigorous mathematical basis to software development [1, p.56]. By using formal methods, software developers can systematically specify, develop, and verify a system [2, p.34]. As formal methods in software development permit more precise specification and earlier error detection [1, p.56], they are been being applied widely in development of safety-critical software systems.

Bowen provides a conclusion of pros and cons of the formal methods [4, p.15]. It states that despite benefits of formal methods, there had been claims about infeasibility of application of these methods for problems in the scale of the real world problems. Sensible proponents of these methods propose that a cost/benefit analysis should be performed before applying these methods and they should be applied only in case of providing apparent advantages in development costs. According to this opinion, using formal methods in development of a simple management information system for a business, as an example, is not worthwhile. Proponents of the formal methods claim that despite the apparent complexity the formal methods add to the process, they indeed reduce the overall cost of software development. Proponents justify it by mentioning the huge cost saving in testing and maintenance, which contain the major software development costs, in comparison with slight increase in cost of specification and design.

Formal specifications are a part of formal methods which use mathematical notation to describe, in a precise way, the properties which a software system must have, without unduly constraining the way in which these properties are achieved [5, p.42]. Mathematical specifications have three virtues: being concise, precise, and unambiguous. Practical experiences show that the mathematical specification of a system is shorter than the English text version as mathematical expressions can convey complexities of real world in short structures [5, p.42]. They are precise because they use mathematical expressions which are precise and accurate. They are unambiguous as mathematical expressions prevent different interpretations from the same expression.

The Z notation is one of the widely-used methods of documenting software specifications in a formal way [21, chapter 11]. Figure 1-1 shows a sample Z schema. Z is a state-based specification language. It considers a software system as an entity which accepts inputs, then may change the internal state according to that and then may provide outputs if required. This vision provides the benefit of isolation from details of implementation like user interface details [5, p.42]. Therefore, one can combine Z specifications together with other forms of specifications documentation methods like UML for one software system. In these combinations the Z-based part of specifications can play the role of describing the core state management part of the system, or maybe just the critical part whereas the other formats can describe requirements for other aspects of the system like user interface.

$AddBirthday$

$\Delta BirthdayBook$
$name? : NAME$
$date? : DATE$

$name? \notin known$

$birthday' = birthday \cup \{name? \mapsto date?\}$

Figure 1-1- A sample Z schema, the smallest units of Z notation. The variables used in this schema are declared on the top of central dividing line and on the part below, the relationship between variables are given.

Because of the formality advantages, some tools are already developed which can transform the formal specifications to code in languages like C++ or Java [6]. As state-based specifications are precise and formal, they can be a good source for estimation of the cost of the software in early development stages of software development process, once the software specifications is in hand.

1.2 Scope

Already in 1987, Samson, Nevill and Dugard, showed that there is a strong and direct influence of specification metrics onto metrics of the implementation [7]. By counting the number of mathematic equations in specifications, the authors demonstrated that an estimation of effort, needed for implementation, is possible. There has been some progress in various aspects of formal specifications since then, however there hasn’t been much progress in the field of cost estimation using recent achievements of formal specifications.

The introduction of specification slicing methods [8] and slice-based specification metrics has opened new windows for measuring properties of specifications with more metrics. For example because of interdependencies between software requirement sections, it had been difficult to measure the quality of specifications. However, by using slicing methods in [9], the author demonstrates that slice-based coupling and cohesion measures in formal Z specifications can reasonably be defined in the same way as in the implemented code.

There are rarely empirically validated correlations between code and specification metrics around. Moreover, for reasons of simplicity in calculation, mostly size-based[1] measures, like number of operations in modules, are used in previous experiments. Therefore, it seems that an empirical study, which investigates relations between measures in Z specification and the implementation measures, can fill up this gap.

The main focus in this thesis work is to examine ***if there is a correlation between formal Z specification measures and implementation related measures***. To answer this question, a measurement experiment is conducted in which the specification and code metrics are measured and the correlation between these measures is investigated using statistical methods. For this purpose, two basic questions are addressed first. The first question is *“which measures are unique descriptors for properties of formal Z specifications?”* The second question is *“which quality and complexity measures, for code or specifications, are used in currently existing predictive*

[1] Categories of formal specifications’ metrics will be discussed in Chapter 2

models?" After addressing these questions, the empirical experiment is conducted to examine *if there is a correlation between formal Z specification measures and implementation related measures.* In concise, this thesis tries to explain the correlation between the measures in specifications and the measures in code, those are input parameters in currently existing estimation models.

1.3 Value

The success of any software development project depends highly on meeting the assigned schedule and budget of the project which are often defined in terms of project plan. Estimation is the basis for planning; planning doesn't make sense without knowing the amount of effort needed for a project. Therefore, having a reliable way of estimating effort needed to perform the tasks is a must for a successful project management. The outcome of this research will provide some help for a reliable estimation for a better plan for at least a part of software projects, those are based on Z specifications.

Regardless of the project and the project management structure, investments are the pushing force for every project because they help to provide the needed resources for the projects. Investment decisions are highly influenced by the schedule and budget of the project which itself is dependent on estimations. Therefore, the expectation of the outcome of this research is to facilitate the decision making process for investment on a part of software projects, those are based on Z specifications.

The Software Engineering Body of Knowledge is sectioned by Key Areas, each of which comprised of sub-areas [10, chapter 1]. The Software Engineering Management key area consists of six sub-areas where the second one, which is Software Project Planning, contains the knowledge about cost estimation. Therefore, the current research will contribute in the cost estimation part of SWEBOK.

1.4 Method

In order to address the main question of this thesis, a set of appropriate[1] metrics applicable on Z specifications are identified. This is achieved by a literature review on existing metrics and the outcome forms the next chapter of the thesis, "Measures in Z Specifications". Then the appropriate code metrics, which work as input for currently existing prediction models, are identified to be measured in the experiment. For this purpose, the prediction models are investigated in a literature review. The result of this study is presented in chapter 3, "Predictive Models."

Then a collection of Z specifications and related implemented code is collected and measured, using the set of provided metrics. Having specification and correspondent implementation measurement values, and using statistical analysis methods, the correlation of these sets of metrics are examined. The result of this section is presented in chapter 4, "The Experiment." The final part of the thesis concludes the results of previous chapters. The following table summarizes the research steps and methods.

[1] The specification-related metrics should have special criteria, which will be defined in chapter 2, to be employable in the experiment

Step	Objective(s)	Method
#1	• Define "key" Z specification metrics. • Collect a set of key metrics.	Literature Review
#2	• Identify the outstanding software cost estimation models. • Identifying the important code metrics for these software cost estimation models.	Literature Review
#3	• Collect a set of specifications in Z and corresponding codes. • Collect the tools for measurement. • Measure them with the collection of metrics. • Examine the correlation between two sets of metrics.	Experiment
#4	• Conclude the important results.	

Table 1-1- Planned steps for the research

One major foreseen risk in this research is shortage in specifications-code pairs. Since many of the software systems based on Z specifications are for safety-critical systems, it's not easy to gain access to their code. As a starting point, parts of specifications and their code from the Tokeneer ID Station[1] software project are available. In the analysis of this risk either of these two approaches are chosen; presenting analysis with less validity or extend the schedule to enlarge the sample.

Another foreseen risk is lack of enough tools for the measurement of all the found key metrics. In this case, measurements of just the metrics for which measurement tools exist or extending the existing tools to cover all metrics are probable solutions. As a starting point a tool which measures a number of specification metrics, namely the size-based measures (CC – conceptual complexity), the structure-based measures (logical complexity and def/use count), and the semantic-based measures (coupling, cohesion, overlap), is available.

[1] http://www.adacore.com/home/products/sparkpro/tokeneer, last visited: January 2011

Chapter 2:
Measures in Z Specifications

2.1 Objectives

The main objective of this section is to provide a collection of specification-related measures which are applicable to Z specifications through reviewing literature of formal specification metrics domain. At the first part of this chapter, the focus area on literature review is specified, then the results of this study are presented, and then the analysis of the resulting metrics is provided. The analysis is from aspect of applicability to the experiment which is conducted later in this research.

2.2 Method

Throughout the literature review the main focus is on the measures which are applicable to Z specifications. Metrics applicable to other formal specifications are applicable to Z as it is a specific formal specification notation. Since this research is aiming at using tools for measuring the metrics in Z specifications, and to keep the right level of abstraction, the mathematical details of explored metrics are kept hidden.

A customized approach similar to the approach explained in [12] was used to conduct the literature review. An empty queue was formed, at the first step, in order to keep track of the list of papers to be read. Then it was populated by the initial set of papers. Bollin's articles ([9], [11]) were used as a starting point for the review and two other papers ([5], [7]) for gaining domain knowledge.

While reading papers, new keywords and concepts related to domain were discovered as well as the papers which seemed indispensable to read for this study. These papers added to the end of the reading queue. The new keywords and concepts are used for narrowing down the search in Google Scholar for related papers. Introduction and conclusion of the selected papers were examined in order to make sure that the paper is in the target domain. Moreover, forward/backward chaining method based on references/citations of papers is used to find more papers [12].

2.3 Formal Specification Measures

As mentioned earlier, Z is a specific formal specification thus all metrics apply to formal specifications, apply to Z as well. The term specification is used instead of formal specification throughout this work for simplicity reasons. However wherever referred to other forms of specification, like text or UML, it's mentioned explicitly.

Bollin in [9] takes the approach of categorizing specifications' metrics into two main categories: complexity and quality metrics. Complexity is defined as "The degree to which the structure, behavior, and application of an organization is difficult to understand and validate due to its physical size, the intertwined relationships between its components, and the significant number of interactions required by its collaborating components to provide organizational capabilities" [3, p.109]. However, the complexity in specifications is usually interpreted to and measured based on attributes which are related to just the size of specifications. One reason is that measures to assess the other attributes of specifications than size, or other qualities of so-called "Good Specifications", had not been defined. It was due to interdependency concepts which were either not at all or only implicitly available for software specifications [9, p.24]. Therefore, Bollin separates complexity from other quality metrics for specifications [9, p.24]. However this categorization seems to be not appropriate as the two categories have serious conceptual overlap.

Specifications of the same size don't have necessarily the same complexity. That's because the relationships between sub-components add more complexity. Therefore, the total complexity is more than what results from just summing up the complexity of sub-components [11, p.158]. Therefore, different sets of metrics to measure other aspects of the complexity are needed. Bollin in [11, p.148] took another approach and categorized the metrics into 3 categories: quantity/size-based, structure-based, and semantic-based. Although the firstly mentioned approach seems to be refined version of the second one by Bollin, the second approach is used throughout this work because of previously mentioned reason.

Quantity/size-based metrics are related to physical size [11, p.148]. These measures are easy to quantify, mostly easy to calculate, and there are lots of studies in this field [11, p.156]. *Lines of specification code*, abbreviated to LOC, is a size-based metric which is measured by counting the lines of specification text [11, p.156]. It's a popular metric because of ease of calculation. However, it's not precise (i.e. value differs if comments or empty lines are counted) [11, p.156]. Samson et al. [7] show that if LOC is defined precisely, which can be done easier in formal specifications than other types, it has a strong correlation with LOC in its implementation code.

A few other metrics, derived from LOC, count primes of a formal specification instead of LOC which have clearer and more comparable semantic complexity [11, p158]. Primes are the smallest structural units of a formal specification. Vinter et al. in [14] show the count of Z specification's structural units correlates with specification's complexity. However, there is no quantitative assessment for that. The approach of counting primes in a specification instead of LOC is called *conceptual complexity* and it provides the ability of comparing and quantifying the complexity of specifications [11, p.163]. Conceptual complexity is a measure for the difficulty of understanding of code/specifications [16, p.73].

Nogueria et al. in [15] define two new metrics, *Fine Granularity Complexity* (FGC) and *Large Granularity Complexity* (LGC). FGC is count of input and output data of specific operation units, called operators. LGC is the summation of number of operators, total number of input and output, and the number of data-types.

Samson et al. [7, p.245] define three metrics, namely *number of equations per operation* (NEQOP), *number of equations per module* (NEQMOD), and *number of operations per module* (NOPS). They also show, in a case study, that these metrics of specifications have correlation with *cyclomatic complexity* of related implementation. Cyclomatic complexity is a complexity measure for code, defined by McCabe [13], which is a measure for a way of modularizing so the resulting modules are both testable and maintainable. It seems important to save a lot of cost of development in testing and maintenance of software.

Kokol et al. in [17] define a metric called *α-metric* for code and they extended it to be applicable on formal specifications. Their case study shows that this metric has different values for the same specifications written with different specifications' languages [17]. There is not much discussion about it after the presentation of this metric and therefore, α-metric didn't find its place in software industry [11, p.158]. Table 2-1 summarizes the size-based metrics with their meaning.

Metric	Conveys
Specifications LOC	Size of specifications in terms of number of text lines.
Conceptual complexity (CC)	Size of specifications in terms of number of primes. A measure for difficulty of understanding of specifications.
Number of operators/equations	Size of specification in terms of the number of operators/equations in a specification/module.
FGC	Complexity of each operator[1] in the system in terms of inputs and outputs.
LGC	Complexity of the whole system in terms of number of operators, input/output data, and types.
α-metric	Measures the information content specifications.

Table 2-1- Size-based metrics for formal specifications

Structure-based complexity metrics have to do with logical and data structures aspect of complexity like the flow of control, number of identifiers and their validity, and the number of references [11, p.148]. Many of the metrics of this category were not applicable until recently. That's because control/data flow is not a dominant aspect of specifications and also formal specifications mostly don't have control structures. Furthermore, it is difficult to generate a control/data flow presentation for specifications [9, p.24]. However, Bollin provides methods for determining data/control dependencies using a graphical representation of specifications called ASRN[2] [11, chapters 4, 5]. An ASRN maps a formal specification to a graph. This mapping allows us to use the vast algorithms and concepts developed for graph theory for the software specifications.

As mentioned earlier, cyclomatic complexity is a semantic-based code-related metric which is defined to measure computational complexity and can be used to measure testability and maintainability of code [13, p.308]. Bollin has provided two metrics by transforming the code-based cyclomatic complexity metric to specifications domain [11, p.165]. Cyclomatic complexity for specifications is calculated by counting all control dependencies in the ASRN of specifications. *Extended cyclomatic complexity*, which is later renamed to *Logical Complexity* by Bollin, is in form of ordered tuple with upper and lower bound values [11, p.166]. The upper bound is the cyclomatic complexity for specifications and the lower bound is calculated by counting vertices with special criteria in the ASRN [11, p.166].

Definition-Use (DU) is a code-based metric which is based on control-flow graph of program [18]. Bollin has provided a transformation of DU for specification domain called DU count [11, p.165]. *DU count* for specifications is equal to the total number of data dependencies in the related ASRN [11, p.165].

[1] Unit of a specific operation

[2] Augmented specification relationship net

Table 2-2 summarizes the structure-based metrics which were discussed in this section.

Metric	Conveys
Logical complexity	Computational complexity of specifications
Definition Use (DU) Count	Data flow dependencies of specifications

Table 2-2- Structure-based metrics for formal specifications

Semantic-based category measures are focused on semantic relationship between sub-components of a component or system and are commonly defined to measure *coupling*, which is a measure for strength of inter-component connections, and *cohesion*, which is a measure for mutual affinity of sub-components of a component [11, p.148].

Carrington et al. define two metrics for specification modules, one for *functional cohesion* and another for *communicational coupling* [19]. These metrics are calculated by counting the state variables of code modules and those which are used commonly between different code modules.

Lakhotia provided a rule-based algorithm to measure cohesion in code by examining the control and data flow of variables [20]. Bollin showed that this measure can, though not fully, be transformed to the domain of specifications using ASRN [11, p.162].

Coupling and coherence metrics are not easily transformable from code domain to specification as these metrics are based on control/data dependencies which are tough to define for specification domain [9, p.24]. However with the methods of specification slicing[1] [8], a few code-based metrics are transformed and applied on slices of specifications, called slice-based metrics.

Bollin [9] provides a transformation for a set of slice-based code-related metrics which measure coupling and cohesion of formal Z specifications using the specification slices. Using these metrics, Bollin shows that we can calculate *coupling*, *cohesion*, and *overlap*. Coupling is a measure for the strength of inter-component connections, and cohesion is a measure for the mutual affinity of sub-components of a component [9]. Overlap expresses the number of primes which are common to all specification slices [9].

Table 2-3 contains a summary of the discussed semantic-based metrics together with their meanings.

[1] For more explanation of static and dynamic slicing using famous Birthday Book sample in Z refer to [8]

Metric	Conveys
Functional Cohesion	Functional cohesion[1] of specifications
Communication Coupling	Coupling of modules of specifications
Rule-based Algorithm	Cohesion (all levels) of specifications
Slice-based Coupling	Strength of inter-slice connections in specifications
Slice-based Cohesion	Mutual affinity of slices of a specification
Slice-based Overlap	The number of primes which are common to all specification slices

Table 2-3- Semantic-based metrics for formal specifications

Now that we have some information for a collection of metrics in hand, we can provide a summary of metrics which are unique descriptors of Z specifications.

Among size-based metrics we identified Specification LOC, Conceptual Complexity, Number of Operators (NEQOP, NEQMOD, NOPS), FGC/LGC, and α-metric. Though Specification LOC is popular because of simplicity of calculation as mentioned before, it can stand for different definitions unless it is defined precisely. α-metric also results in different values while measuring different specifications with different languages written for the same functionality. Therefore, it can not be a good candidate for specification-based estimations. The conceptual complexity is easy to calculate and provides ability to compare since it is based on concrete formal specifications' units.

For the structure-based metrics category we identified Cyclomatic Complexity, which will be referred to as Logical Complexity hereafter, and Definition-use count metric. Both these metrics are calculable and precise as they have mathematical-related definitions and based on ASRN which itself is based on the graph theory.

For semantic-based metrics we identified functional cohesion, communicational coupling, rule-based approach, and slice based coupling, cohesion, and overlap. As mentioned, the rule-based approach for code is not thoroughly transformed for specifications and it can not be used as a reliable metric for specifications. No specific drawback is found for the rest of metrics in semantic-based category. Table 2-4 summarizes the metrics explored in this study.

[1] All parts which contribute to a single and specific function [11, p.154]

Cat.	Metric	Comments
Size-Based	Specifications LOC	Not precise but measurement tools are available.
	Conceptual complexity (CC)	No drawback found and measurement tools are available.
	Number of operators/equations	No drawbacks found but no measurement tools available
	FGC/LGC	No drawbacks found but no measurement tools available
	α-metric	Different values for different languages
Structure -Based	Logical complexity	No drawback found and measurement tools are available.
	Definition Use (DU) Count	No drawback found and measurement tools are available.
Semantic-Based	Functional cohesion	No drawbacks found but no measurement tools available
	Communicational coupling	No drawbacks found but no measurement tools available
	Rule-based approach	Not thoroughly defined for specifications
	Slice-based coupling, cohesion, and overlap	No drawback found and measurement tools are available.

Table 2-4- Summary of metrics for Z specifications

Some of the metrics, like Lines of Code, have different interpretations and measurement methods. Therefore, such metrics should be defined precisely together with the measurement method in case of using in an experiment. For this reason, the precise definition and measurement method for the metrics used in experiment is provided in chapter 4 which explains the experiment details.

Chapter 3:
Predictive Models

3.1 Introduction

The main objective of this chapter is to present the result of investigation in the salient software cost estimation models which currently exist and are already validated in practice. The investigation performed with the focus on the main advantages and drawbacks of these models and their connection points to this thesis in terms of code or specification metrics. Therefore, the goal of this short study is to find at least one reliable cost estimation model for which a code or specifications metric is an important input.

Defined by Wikipedia[1], "Cost estimation models are mathematical algorithms or parametric equations used to estimate the costs of a product or project." Software cost estimation techniques are used for a number of purposes including budgeting, trade-off and risk analysis, project planning and control, and software improvement investment analysis [22, p.177]. As "effort" and "cost" are in a direct relation in software projects, cost estimation and effort estimation terms are sometimes used in each other's place.

This short review has been performed in order to find the models which suit the purposes of this thesis. These models should be reliable; means that they should have been empirically validated. They should also have inputs from code or specification metrics so that a relation can be made to the output of the later experiment of this thesis.

To reach to the outstanding papers in this topic for the review, a systematic method is applied. At first, Google Scholar is used for the search using the following logical combination of keywords:

"Software" AND "estimation" AND ("cost" OR "effort")

Then the abstract, introduction, and conclusion of the resulting papers were examined to assure that they're relevant to the purposes of the study. Number of citations, publish date, and references of papers are also examined in order to prioritize them. The two next sections will provide the results of reviewing the selected set of papers on software cost estimation models.

3.2 Cost Estimation Approaches

A classification for estimation models seems necessary in order to present the results of the review in an organized way. One of the major differences between estimation models is based on using Source Line of Code (SLOC) as the primary input for the model [22, p.417]. This approach provides a simple categorization of the models; those models which use SLOC as an input and those which do not. The models which don't use specification or code metrics are not in the focus of this review as they can not be integrated into this thesis to form a total cost estimation model.

Boehm provides six approaches of estimation techniques, namely model-based or parametric, expertise-based, learning-oriented, dynamics-based, regression-based, and Composite [22, p.178].

[1] Last visited: February 2011

Another estimation approach categorization is provided by Jørgensen et al. in which they have identified 13 estimation approaches [28, p.42]. However, they have used a simple top level categorization, with 3 categories, implicitly throughout the text: expert estimation, formal estimation, and combination-based estimation [28, p.39]. Since the expert- and combination-based estimation techniques can not be related to this thesis' results, they won't be in focus.

3.3 Estimation Models

SLIM[1] is a software life-cycle model which used a Rayleigh manpower distribution model for estimating the needed effort for a software project [24]. A Raylegh curve is the graphical presentation for a mathematical equation which shows the relation between delivery time and needed effort for a software project. SLIM is a parametric model and can be calibrated using finished projects data or by answering a set of questions in case of lack of previous data [22, p.179].

According to SLIM's cost estimation formula, a project cost can be reduced to 50% by simply increasing its schedule by 19% [25, p.10] which seems far from the real world software projects data. This issue made a validity weakness for this model. Nevertheless, SLIM has a good performance when it is compared to a few other outstanding estimation methods [23, p.428]. SLIM is a proprietary model and therefore, it has a limitation for using this method for cost estimations.

Doty is another parametric cost estimation model which considers a number of characteristics of software projects as factors in its cost estimation formula [25, p.12]. Estimation formula in Doty has a discontinuity when code size, as input parameter, is equal to 10K delivered source instructions. As another weakness, the estimated cost increases by 92% by simply answering "yes" to one of characteristic factors [25, p.12].

COCOMO II is an updated version of the COnstructive COst MOdel, the popular cost estimation model of the 1980s [22, p.189]. COCOMO II covers the weaknesses of the old version in confronting the new software development processes and capabilities [22, p.189]. The initial version of the model consists of three sub-models each of which has their own application area; the application composition model for the software projects which uses ICASE[2] tools for rapid application development; the early design model is aimed at early cost estimation in projects and accepts source lines of code (SLOC) or function points[3] as the main input together with 5 scale factors and 7 effort multipliers; the post-architecture model is applicable when the top level design is complete and it accepts source lines of code or function points as the main input together with 17 effort multipliers and 5 scale factors [22, p.190]. No specific drawbacks are found for COCOMO II in the reviewed papers.

Mulisek et al. in [26] have provided an analysis on sensitivity of COCOMO II model. Their research reveals that the COCOMO II model is sensitive firstly to size input parameter and then to effort multipliers. Therefore, the experiment of this thesis which is aimed at providing a precise estimation for the size of code, as input parameter of the model, can help to provide more precision for COCOMO II model. The internal equations and parameter values are also fully available for this model. Therefore, this model seems to be good candidate to be related to the results of the experiment in this thesis in order to form a total cost estimation model.

PRICE-S is a parametric and proprietary estimation model which has been used in several U.S. DoD, NASA, and other government software projects [22, p.182]. Since the model equations are not published, it can not be used for this research purposes.

[1] Software Lifecycle Management

[2] Integrated Computer Aided Software Engineering

[3] "A function point is a unit of measurement to express the amount of business functionality an information system provides to a user.", Wikipedia, last visited: February 2011

There are a few other estimation techniques, like Checkpoint, ESTIMACS, SEER-SEM, and SELECT, which are based on functionality-based size measures or other OO-related metrics [22]. OO-related measures are not in hand at least until the early design stage since they are dependent on the architectural and design decisions. Since functionality-based size metrics, like function points, are not the dominant aspect of formal specifications, this thesis results are not beneficial for them. Therefore, these models are not reviewed in this study.

Table 3-1 summarizes the advantages and drawbacks of the candidate models for a total cost estimation model based on the later experiment in this thesis.

Model	Advantage(s)	Drawback(s)
SLIM	-Good precision	-Proprietary model
Doty	-Easy to calibrate	-Discontinuity on DSI=10K -Lack of sufficient precision
COCOMO II	-Applicable in different stages of SW life-cycle -Easy to calibrate	No drawback found in reviewed papers
PRICE-S	-Used in government projects	-Proprietary

Table 3-1- Advantages/drawbacks summary of reviewed cost estimation models

Briand et al. in [27] provided an analysis on accuracy of software cost estimation models. The results of their research show that the estimation models which are based on analogy are less accurate than the rest. With this exception all other cost estimation models have more or less the same accuracy. Another research reveals that the algorithmic estimation techniques should be calibrated in target organizations to work well [23, p.427]. Moreover, it should be mentioned that there’s no single cost estimation model which can suit for all situations [22, p.177].

3.3.1 Input Parameters

SLIM uses Delivered Source Instruction (DSI) as the main input prameter which is a metric for describing the size of code. Boehm defines DSI as program instructions created by project personnel that are part of the final product [23, p.418]. DSI can be assumed as a more precise definition for source lines of code which doesn’t include comments, empty lines, and etc. There are a few tools[1] which can calculate DSI in a variety of programming languages. Other input parameters in basic model of SLIM consist of development time and a technology constant which can be calibrated based on past projects [25, p.10].

Similar to SLIM, Doty also uses DSI as one of its input parameters. The other input parameters include the factors for characteristics of software projects. These factors accept the value of Zero or One according to the description of factor and therefore, they are not to be estimated.

As mentioned before, different sub-models of COCOMO II use a variety of input parameters from which the number of source lines of code (SLOC) is estimable. The rest of input factors are either determined parameters, like function points, or parameters related to the characteristics of the project which are not to be estimated.

Three different code size metrics, namely Count Line Code, Lines Executable, Lines Declarative, are considered for the experiment of this thesis. Using these three

[1] http://www.locmetrics.com/alternatives.html, last visited: February 2011

metrics, one can calculate the value for various definitions of SLOC and DSI. Table 3-2 summarizes these metrics with their definitions.

Metric	Definition
Count Line Code	Is equal to Lines Executable + Lines Declarative
Lines Executable	total lines that have executable code on them
Lines Declarative	total lines that have declarative code on them

Table 3-2- Code metrics to be considered in the experiment

3.4 Summary

According to the results of this study, SLOC and DSI are the most commonly used metrics in code which are used as input for estimation models. SLOC and DSI are quantifiable and objective, though difficult to estimate at the beginning of a software project [22, p.417].

Estimation of size of the software, in terms of source lines of code, seems to be the common problem for models which have such an input. Therefore, regardless of the discussed cost/effort estimation models, the results of the experiment in this thesis can be used in every model which uses SLOC or DSI as an input for the size of the software.

The next chapter provides details of an experiment which is conducted in order to investigate the correlation between measures in Z specifications and measures in implemented code of a software system.

Chapter 4: The Experiment

4.1 Introduction

The study conducted in chapter two revealed that specifications in Z can be measured with different types of metrics. The results of literature review, in the previous chapter, also show that there are reliable software cost estimation models which need the SLOC or DSI as input parameter. Therefore, a study which investigates the correlation between metrics of Z specifications and metrics of code can provide a means to estimate total software cost once the specifications are in hand.

The next section sets the design of the study. The later sections state the results and make a discussion over those results. This chapter will end with an analysis of the threats to the validity of this study.

4.2 Methodology

4.2.1 Subjects

Subjects for this study are a set of pairs of code modules together with their related specifications in Z. In order to collect the sample, by searching on web and sending emails to a few major researchers in Z formal specifications field, it revealed that there is just one software system in industrial scale whose both the code and Z specifications are publicly available and it is called Tokeneer ID Station[1], implemented via ADA programming language. However, there are many subjects of Z specifications with code for the learning purposes which could not be used since this experiment is focused on industrial-scale real world problems.

A software module can be considered as a set of instructions which accepts inputs, performs the computations, and probably changes the state and/or generates outputs. With this definition, one software system can be broken up into several software modules, each of which can be considered as a subject for the study. However, the main issue here is to find the modules in a proper level of granularity.

A code module should fit in a part of specifications to be the good representative of the specifications. It means that the module should implement exactly that part of specification, neither more nor less. Figures 4-1 and 4-2 depict a particular example of this situation. In this example a utility module which is providing different services for several modules cannot be a part of one of the subjects (figure 4-1). That's because it is providing some other features for other modules which are not in a particular subject. However, if a related specification slice exists for the utility module, it can be a subject itself (figure 4-2).

Because of the mentioned issues in providing subjects, the code and documentations of the Tokeneer are investigated precisely and in different abstraction

[1] www.adacore.com/tokeneer , Last visited: March 2011

levels of code, and subjects are identified one by one. Therefore, all the subjects for this study are formed via a step-wise procedure which is explained here in this section.

According to code documentations of Tokeneer, most of the procedures are mapped to one or a few formal design traceability units. A formal design traceability unit is a package of formal design schemata in Z. However, they are not specification schemata, to be measured, and they just provide a means to trace to the formal specifications traceability units. Formal specifications traceability units are packages containing the specification schemata in Z which are to be measured. INFORMED Design[1] document of the Tokeneer project is used to trace the procedures in code which lack the traceability documentations.

Regarding the situation for extracting subjects, each final subject consists of a cluster of procedures in code together with a cluster of related Z schemata. Therefore each subject contains a set of Z schemata and the set of code procedures which implement those schemata. To keep the traceability, a table is formed with four columns: Procedures, Formal Design (FD) Traceability Units, Formal Specifications (FS) Traceability Units, and Z schemata.

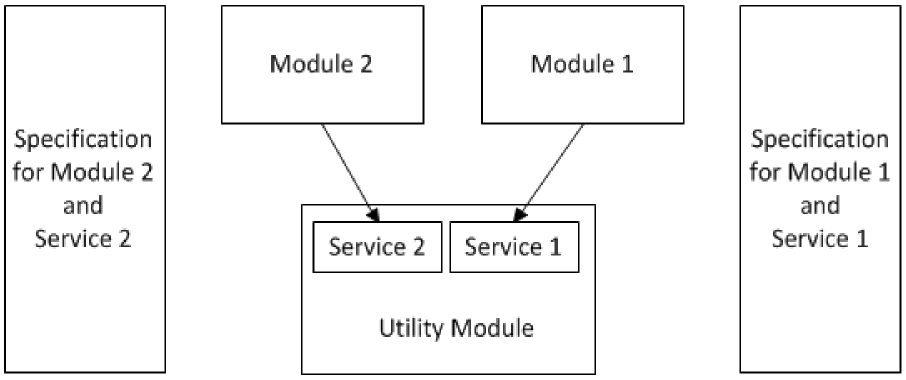
Specification for Module 2 and Service 2
Module 2
Module 1
Service 2
Service 1
Utility Module
Specification for Module 1 and Service 1

Figure 4-1- A non-mappable situation which results in no sample

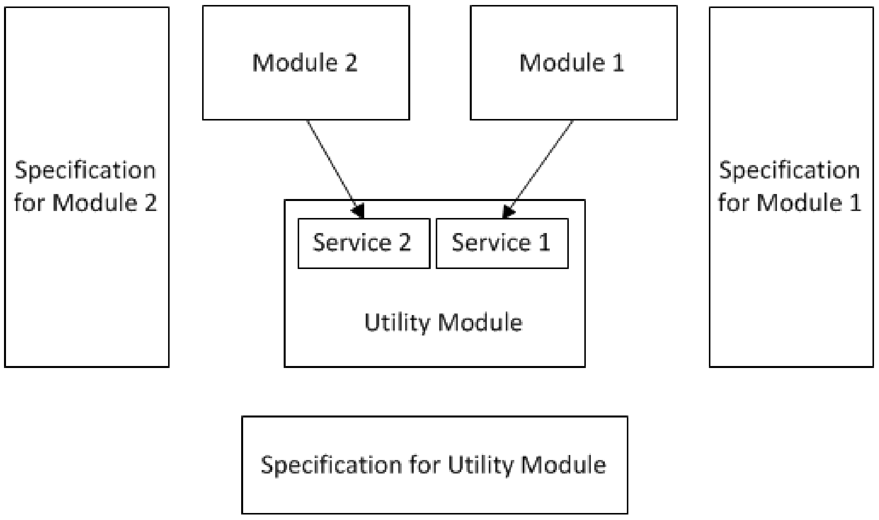
Specification for Module 2
Module 2
Module 1
Service 2
Service 1
Utility Module
Specification for Module 1
Specification for Utility Module

Figure 4-2- A mappable situation which results in 3 samples

[1] http://www.adacore.com/wp-content/files/auto_update/sparkdocs-docs/Informed.htm, Last visited: September 2011

The sample extraction procedure starts with choosing one procedure of code and listing it under the column for procedures in a clean table for a new sample. FD units for the chosen procedure are listed under the FD column. Under the FS column, the FS units related to the FD units are listed in the same manner. To this point of process the list contains just the FS units related to the primarily chosen procedure. However, there may be still some procedures which participate in implementing the listed FS units. Therefore, another scan in reverse way is performed.

In this way, the list of FD units is enriched by finding all FD units related to the list of FS units. Then again the code is inspected for other procedures which relate to the listed FD units. This forward/backward procedure is performed until no more entry can be found and added to the lists.

At this point one subject is formed containing the list of procedures and the list of Z schemata related to the FS units. It's good to mention that the subjects with loosed traceability are eliminated since their code and specification clusters are not representing each other properly. A total of 28 subjects are formed via this procedure and they are listed in Appendix A.

4.2.2 Variables

The independent variables in this study are the metrics in specifications and dependent variables are the code metrics. These specification and code metrics are chosen through procedures described in the previous chapters and they are defined precisely here in this section. Study subjects are measured with these metrics and form the variable values. Table 4-1 shows the metrics with which the Z specifications are measured. An exact and clear definition is also provided to remove the ambiguity so that the experiment becomes repeatable. For the calculation of Z specification measures, an Eclipse plug-in from the ViZ project is used [29].

Cat.	Metric	Definition
Size-Based	Specifications LOC	Number of text lines in the specifications.
	Conceptual complexity (CC)	Number of primes in the specifications.
Structure-Based	Logical complexity	In the ASRN of the specification: Edges - Nodes + Connected Components
	Definition Use (DU) Count	Number of data dependencies in the ASRN of the specifications.
Semantic-Based	Slice-based Coupling	According to Bollin's paper [9, p.26], it is calculated as the amount of information flow between schemas.
	Slice-based Cohesion	According to Bollin's paper [9, p.26], it's calculated via Tightness and Coverage metrics
	Slice-based Overlap	The number of primes which are common to all specification slices

Table 4-1- Specification metrics and measurement methods

Table 4-2 lists the code metrics to be measured in the experiment together with the clear definition of them. The metrics are chosen according to the results of the study in chapter three. The cyclomatic complexity metric is added to this list in order to investigate the correlation between the metrics in specifications with the complexity of code. If this correlation is found, it helps to pre-locate the parts of the system with high complexity in order to take special considerations in implementation. The metrics in code are calculated using a tool called SciTools Understand[1] for which a temporary license is acquired from its producer company.

Metric	Definition
Count Line Code	The number of lines that contain source code. Note that a line can contain source and a comment and thus count towards multiple metrics. For Classes this is the sum of the Count Line Code for the member functions of the class.
Lines Executable	total lines that have executable Ada code on them
Lines Declarative	total lines that have declarative Ada code on them
Cyclomatic Complexity	Cyclomatic complexity [13] In the control flow graph of the code: Edges - Nodes + Connected Components. This metric is applicable just in procedure level

Table 4-2- Code metrics and measurement methods

4.2.3 Hypotheses

There are two hypotheses in this study; there is no correlation between selected metrics in Z specifications and metrics in code of software systems or there is a correlation between them. Therefore it's assumed that the metrics in Z specifications have absolutely no effect on the metrics in code unless a reason is found to reject this hypothesis. The hypotheses are formulated as follows:

- Null hypothesis (H_0): Selected metrics in Z specifications *do not* correlate with metrics in code for a software system.
- Alternative hypothesis (H_1): Selected metrics in Z specifications correlate with metrics in code for a software system.

4.3 Results

As mentioned before, each subject of this study contains a set of procedures together with a set of Z schemata. Therefore, the main aim is to calculate the mentioned metrics for each subject, not for each procedure and schemata in the subjects. Hence, these metrics should be summarized for each subject.

According to the concepts of size and complexity, the size and complexity of a group of procedures is equal to summation of size and complexity of each procedure in the group. Therefore, it's enough to calculate the summation of the count line code, lines executable, lines declarative, and cyclomatic complexity of all procedures in a particular subject to achieve the values of these metrics for that subject.

The size and complexity metrics of Z schemata are calculated for each subject in the same manner. However, the calculation is not simple for the sematic-based measures in Z unlike the other measures. One simplistic way of calculating semantic-based measures for a group of schemata is to calculate the average of the values of metrics. The results of the measurement of metrics for every sample together with a summary table for all the samples are provided in Appendix B.

[1] http://www.scitools.com/index.php, Last visited: March 2011

4.4 Discussion

The main aim in this correlational study is to search for a reason to reject the null hypothesis of the study. Therefore, a statistical reason should convince that there is a correlation between one or more independent variables and the dependent variables. For this purpose, regression test is applied to the measurement data. The analysis results are discussed here in this section.

According to the regression analysis concepts, if the P-value for an independent variable is less than 0.05, it means that there is less than 5% chance of the dependent variable values would have come up in a random distribution [30]. In other words, there is a probability of 0.95 that the independent variable affects the dependent variable and hence, the null hypothesis is rejected. Therefore, each and every metric in code, or the same dependent variable, is investigated to find such a correlation.

Table 4-3 has a summary of regression analysis results. According to this table and for Count Line Code as the dependent variable, the P-values for a few of independent variables, or the same specification metrics, are less than 0.05. These metrics are namely Specification Line of Code, Conceptual Complexity, Definition-Use, Minimum Coverage, and Coupling. Therefore the null hypothesis is rejected for these specification metrics and they are in correlation with Count Line Code. Regression test results for Lines Executable as dependent variable indicate that Specification Line of Code, Conceptual Complexity, Definition-Use, Minimum Coverage, and Coupling metrics in specifications have correlation with Lines Executable in code. Comparing to Count Line Code and Lines Executable, there are fewer specification metrics, namely Definition-Use, Minimum Coverage, and Coupling in correlation with Lines Declarative. Unlike the other code metrics, Cyclomatic Complexity in code doesn’t show any correlation with metrics in specifications, even Cyclomatic Complexity of specifications.

The R-Square value for the regressions shows the percentage of variation of code metrics which is explained by metrics in specifications. In other words, the value of R-Square indicates whether a regression equation is useful to predict the value of a specific metric in code from metrics in Z specifications [31, p.240]. Therefore, 85% of variation of Count Line Code is explained by the metrics in specifications. This amount grows to 88% for Lines Executable but it is weaker for Lines Declarative which is 73%. For Cyclomatic Complexity in code, just 44% of variation of the metric is explained by specification metrics.

The value of Significance F indicates the amount of reliability of regression results. If Significance F for a regression analysis is less than 0.05, it proves that regression analysis results are reliable. Therefore, results of regression analysis are highly reliable for Count Line Code and Lines Executable, and reliable for Lines Declarative. However, the results are not reliable for Cyclomatic Complexity in code.The detailed results of regression analysis for metrics in code are provided in Appendix C.

Regression Parameter		Count Line Code	Lines Executable	Lines Declarative	Cyclomatic Complexity
R-Square		0.85	0.88	0.73	0.44
Significance F		0.00009	0.00002	0.007	0.38
P-Vale	LOC in specifications	0.03	0.01	0.19	0.52
	Conceptual Complexity	0.02	0.01	0.11	0.21
	Cyclomatic Complexity Low	0.26	0.24	0.14	0.21
	Cyclomatic Complexity High	0.23	0.18	0.15	0.18
	Definition-Use Count	0.04	0.02	0.05	0.06
	Tightness	0.37	0.34	0.39	0.68
	Min Coverage	0.05	0.05	0.04	0.28
	Coverage	0.20	0.20	0.11	0.26
	Max Coverage	0.60	0.63	0.35	0.41
	Overlap	0.35	0.44	0.30	0.83
	Coupling	0.02	0.02	0.03	0.53

Table 4-3- Summary of regression analysis results

Figures 4-3 to 4-6 are Scatter plots that show the deviation of regression values from real values. X dimension for all the points in these figures is the regression values. However, Y dimension for points are in two types. For the points with + shaped markers, Y is regression values and for the points with diamond-shaped markers, Y is real values for the metric in code. Therefore, the regression values form the line Y=X and the real values have some deviation from regression values in Y dimension.

Figures 4-3 implies that except a few outliers, the rest of points have acceptable deviation from regression values for Count Line Code. For Lines Executable, the points are more integrated in figure 4-4. However, figure 4-5 shows a more intense scatter for Lines Declarative and this is even more intense for Cyclomatic Complexity of code in figure 4-6.

Figures 4-7 to 4-10 project the deviation of regression values from real values for each sample. The X-axes are the sample numbers. These figures confirm the regression results which show there can be a reliable estimation for Code Line Code, Lines Executable, and Lines Declarative based on metrics in specifications. However, the deviation of estimated values and real values for Cyclomatic Complexity of code is rather intense.

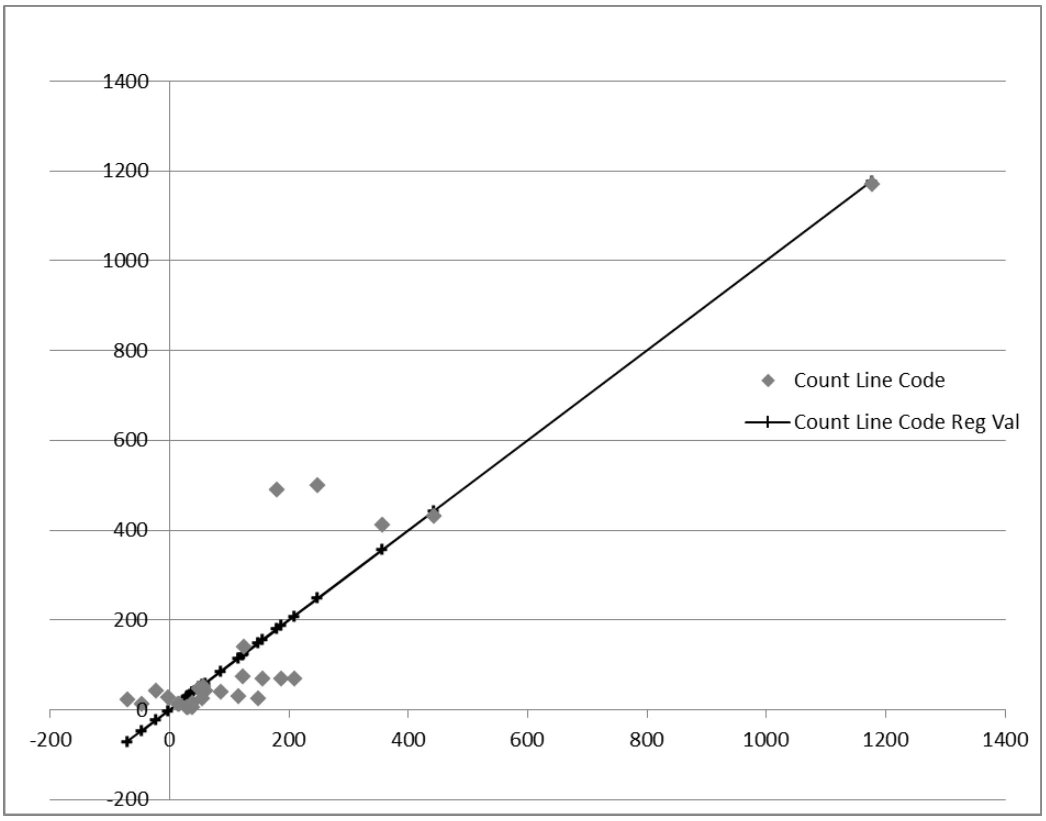

Figure 4-3- Scatter plot based on regression analysis results for Count Line Code

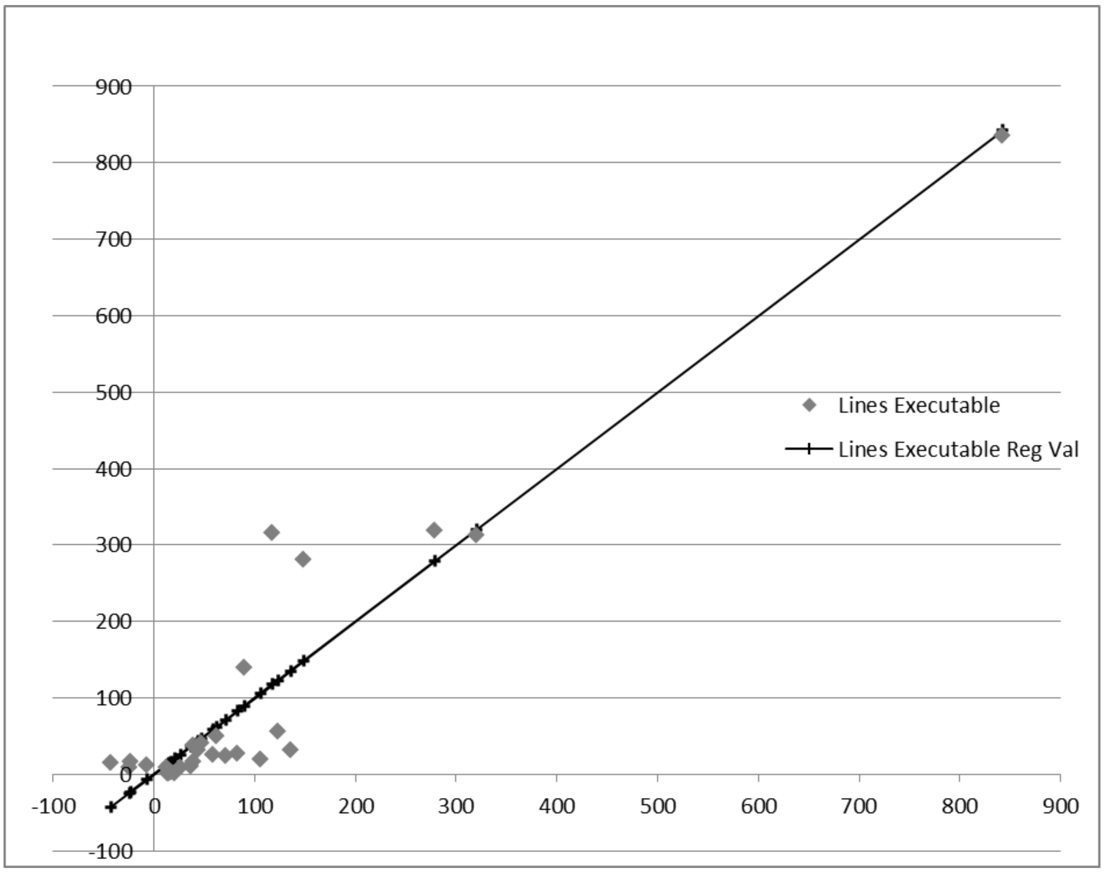

Figure 4-4- Scatter plot based on regression analysis results for Lines Executable

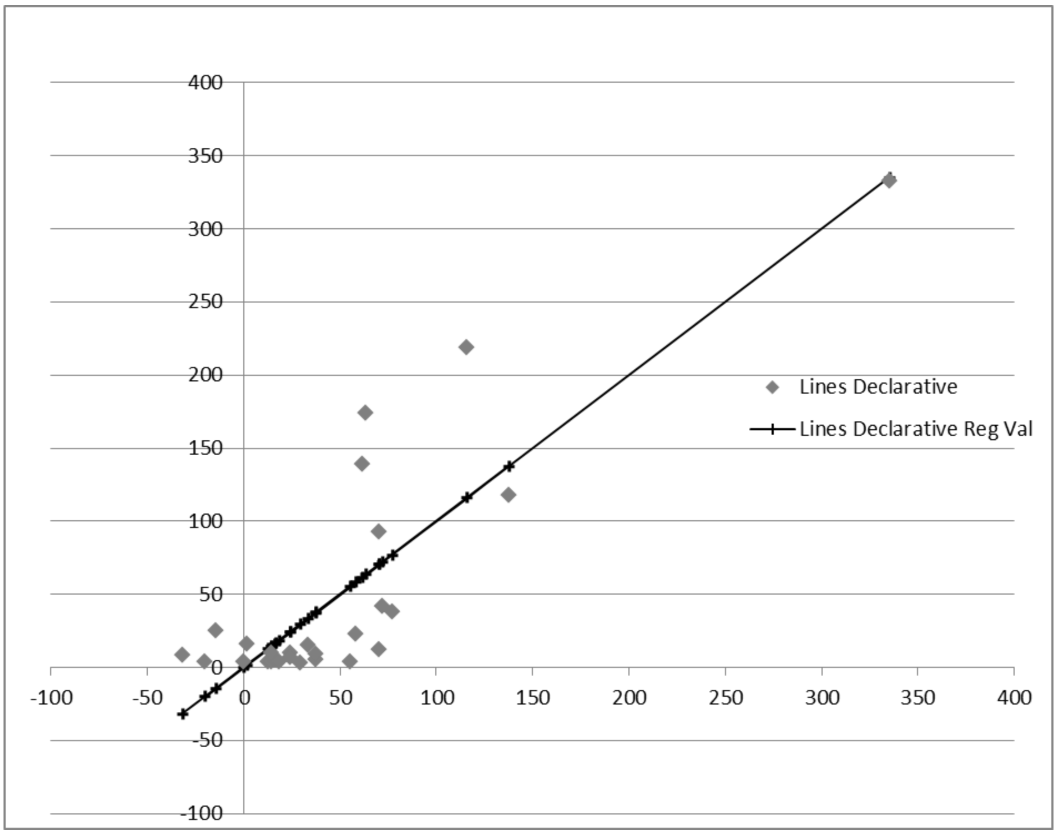

Figure 4-5- Scatter plot based on regression analysis results for Lines Declarative

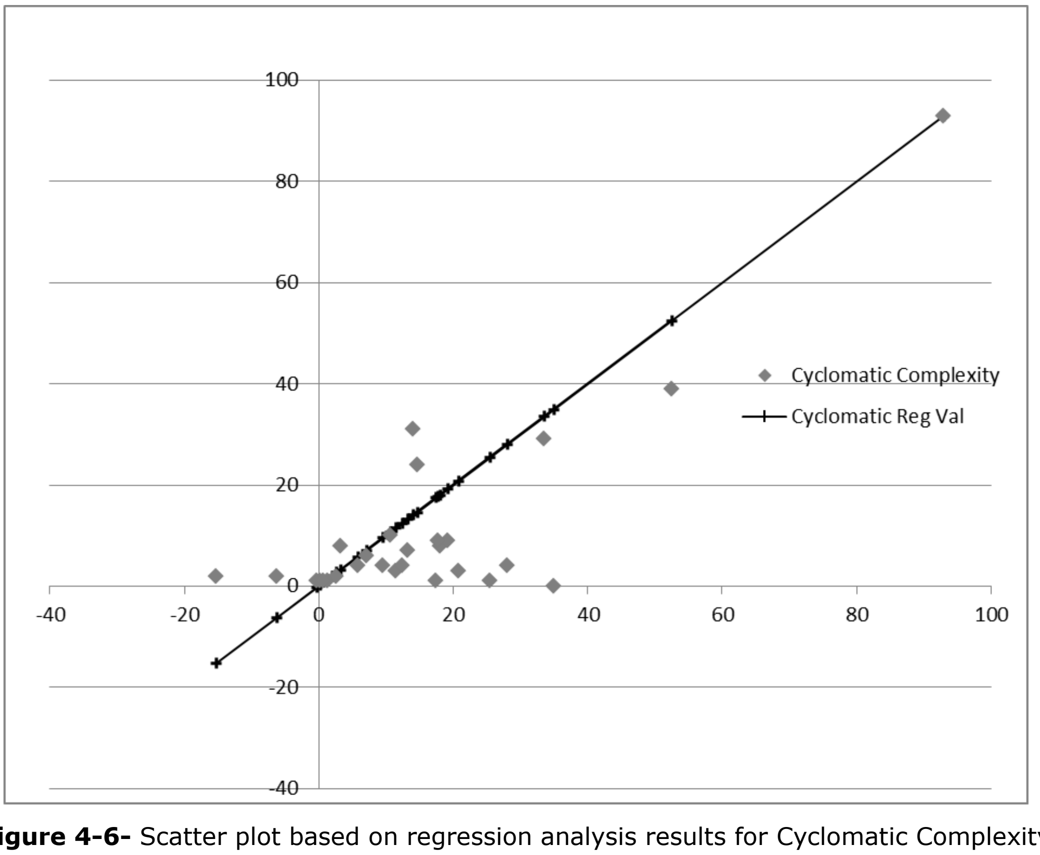

Figure 4-6- Scatter plot based on regression analysis results for Cyclomatic Complexity

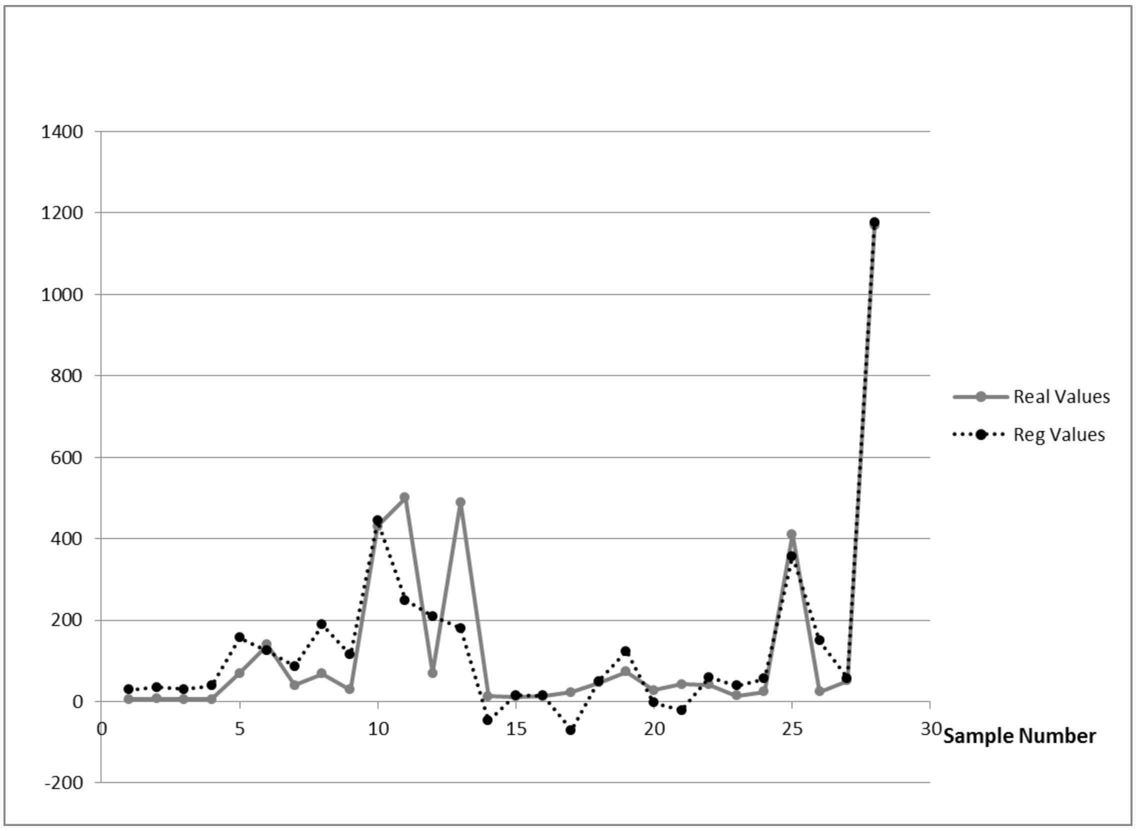

Figure 4-7- Real values vs. regression values for each sample for Count Line Code

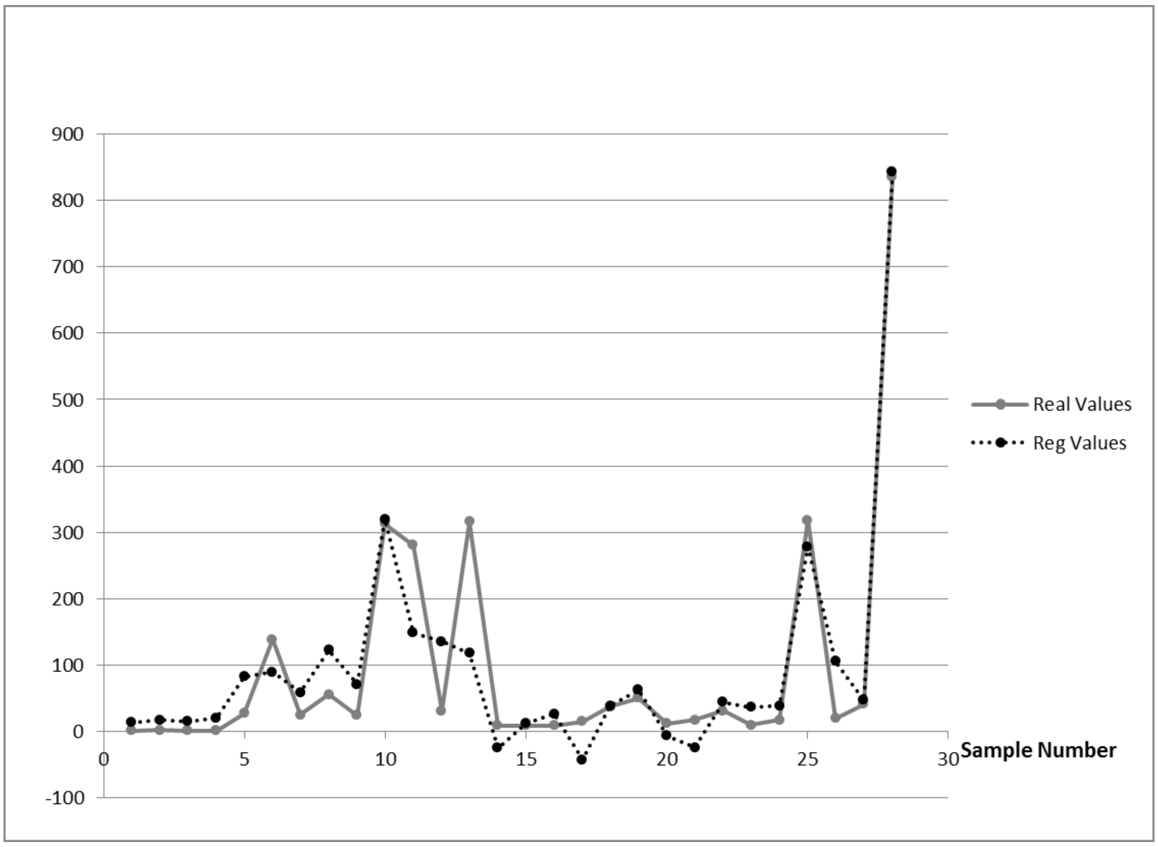

Figure 4-8- Real values vs. regression values for each sample for Lines Executable

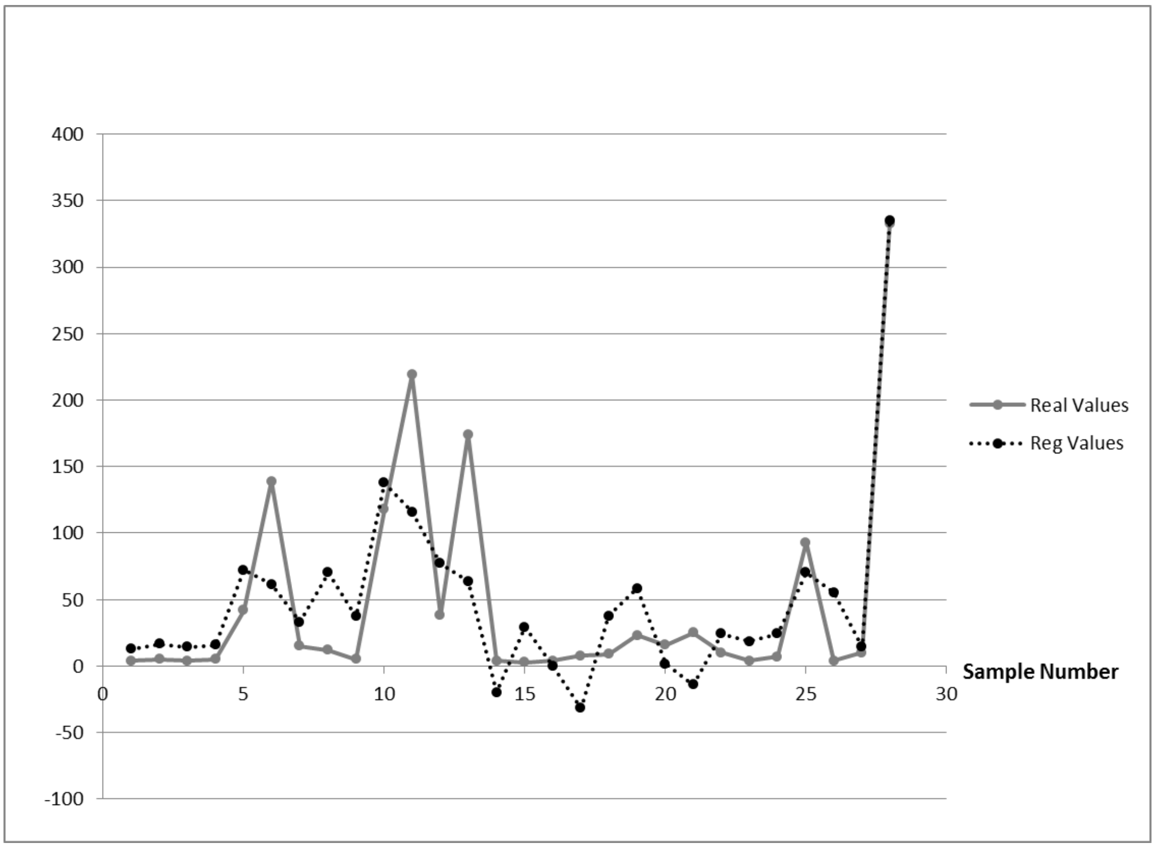

Figure 4-9- Real values vs. regression values for each sample for Lines Declarative

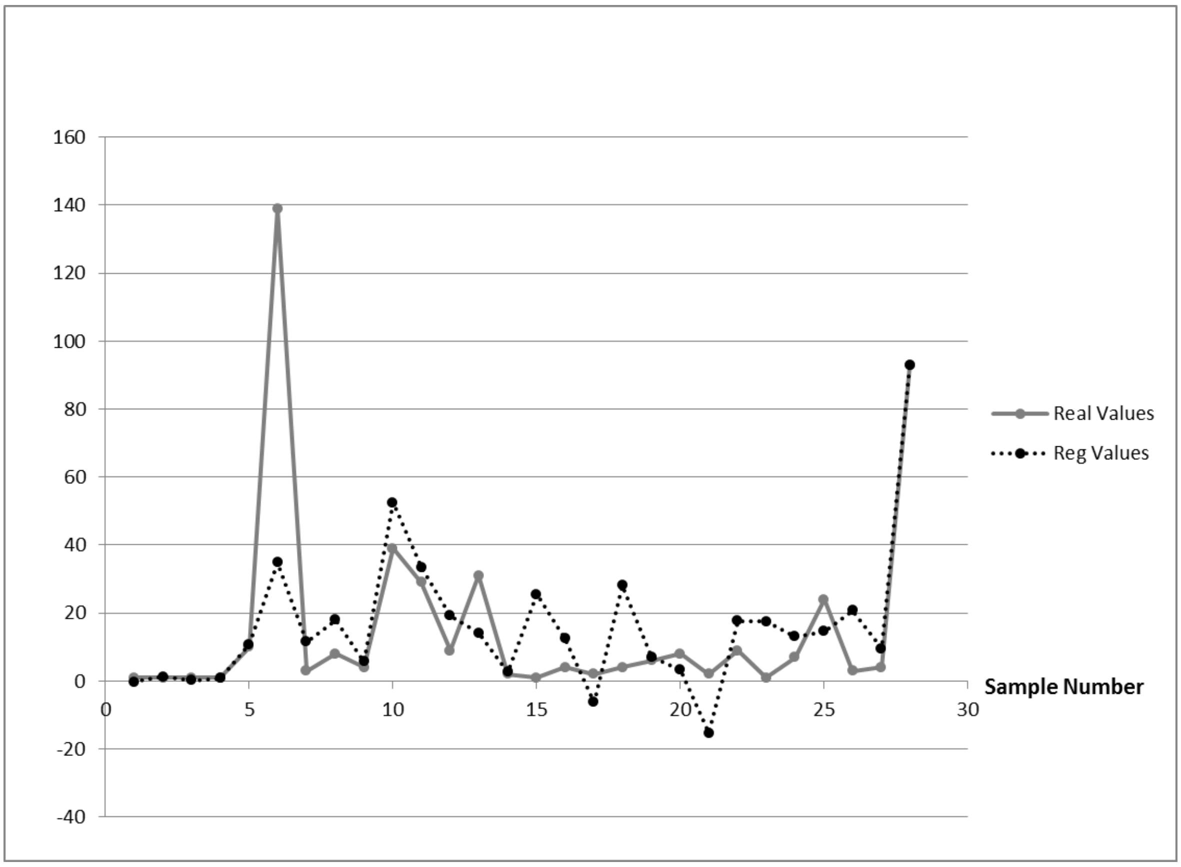

Figure 4-10- Real values vs. regression values for each sample for Cyclomatic Complexity

Regression analysis results together with the graphs confirm that three out of four chosen metrics in code are predictable from metrics in Z specifications. Table 4-6 presents the coefficients for these metrics in regression results.

Component	Count Line Code	Lines Executable	Lines Declarative	Cyclomatic Complexity
Intercept	120.21	60.74	56.01	5.78
LOC in specifications (LOCS)	-1.79	-1.35	-0.44	-0.12
Conceptual Complexity (CC)	3.26	2.48	0.97	0.40
Cyclomatic Complexity Low (CCL)	4.89	3.14	2.81	1.27
Cyclomatic Complexity High (CCH)	-0.03	-0.02	-0.02	-0.01
Definition-Use Count (DU)	-0.18	-0.13	-0.07	-0.04
Tightness (TI)	566.57	377.12	231.53	60.40
Min Coverage (NCOV)	-1473.22	-910.95	-673.96	-180.65
Coverage (COV)	1076.74	669.17	579.60	218.94
Max Coverage (XCOV)	-261.73	-152.67	-205.97	-98.11
Overlap (OLAP)	194.55	99.66	94.01	10.30
Coupling (COUP)	-1004.59	-605.06	-389.66	-58.85

Table 4-6- Coefficients for the components of regression analysis

According to the presented discussions, the formula for predicting chosen metrics in code is presented here.

$$\begin{aligned} &\mathit{Count\ Line\ Code} \\ &\quad = -1.79\,LOCS + 3.26\,CC + 4.89\,CCL - 0.03\,CCH - 0.18\,DU + 566.57\,TI \\ &\quad - 1473.22\,NCOV + 1076.74\,COV - 261.73\,XCOV + 194.55\,OLAP - 1004.59\,COUP \\ &\quad + 120.21 \end{aligned}$$

$$\begin{aligned} &\mathit{Lines\ Executable} \\ &\quad = -1.35\,LOCS + 2.48\,CC + 3.14\,CCL - 0.02\,CCH - 0.13\,DU + 377.12\,TI \\ &\quad - 910.95\,NCOV + 669.17\,COV - 152.67\,XCOV + 99.66\,OLAP - 605.06\,COUP + 60.76 \end{aligned}$$

$$\begin{aligned} &\mathit{Lines\ Declarative} \\ &\quad = -0.44\,LOCS + 0.97\,CC + 2.81\,CCL - 0.02\,CCH - 0.07\,DU + 231.53\,TI \\ &\quad - 673.96\,NCOV + 579.60\,COV - 205.97\,XCOV + 94.01\,OLAP - 389.66\,COUP + 56.01 \end{aligned}$$

As a conclusion from the results presented in this chapter, it can be said that apart from Cyclomatic Complexity of code, there are signs for the rest of code metrics, namely Lines of Code, Lines Executable, and Lines Declarative, which show there are correlations between selected metrics in specifications and those code metrics. Nevertheless, there are some validity threats to the results which prevent this study to make any claim about the precise quality of this correlation. Therefore, a few other statistical tests, which could be applied in order to identify the precise quality of this correlation, are ignored in this thesis work. These validity threats are discussed in the next section.

4.5 Threats to Validity

Two applied simplifications during this research might make threats to validity of the results. These simplifications have been applied during extraction of the 28 samples and also, during measurement of metrics for each sample.

The first simplification has been applied in order to form subjects of the study where the parts of code should represent the implementation of the parts of specifications. In the process of extracting samples, which is explained before, an approximation technique is applied. The quality and reasons for applying this technique is explained here.

Procedure call-backs in code are usual in almost all programming languages and styles, including the code which is investigated in this study. Though the sub-procedures of a cluster of procedures in a particular sample participate in implementation of the cluster of specifications of that sample, they are not taken into account in this study. That is because considering one further level of procedure call-backs will lead to cluster interlacement until whole the code becomes just one huge sample. This interlacement happens because of some procedures which are called by two or more procedures from different code clusters. Therefore the code parts approximately, and not precisely, represent the implementation of the specifications and this is a threat to validity of results of this study.

The other threat to validity of results is in the way of calculating metrics for samples which is explained in section 4-3. As each sample of this study consists of a cluster of specifications and a cluster of procedures in code, the metrics should be calculated for whole the cluster rather than one particular procedure or Z schemata. There are problems with calculating the metrics since the metrics are defined for a single procedure or Z schemata. In order to get around this problem it is needed to extend the definition of those metrics for clusters.

According to definition of size and complexity metrics, The value of a size or complexity metric for a cluster of items, either procedures in code or schemata of Z specifications, is equal to summation of the values of that metric for each of items in that cluster. For example the value of Count Line Code for a cluster of procedures is equal to summation of values of Count Line Code for each of procedures in that cluster.

For semantic-based metrics in Z specification the average of measurement values for each schema in the cluster is calculated. This is the simplified way of calculating these metrics, though compatible with the definition of the metrics. The more precise way of calculating semantic-based metrics for clusters is more complex and in that way, the weight of each schema in the cluster should be taken into account. To find the weight of each schema, the structure of schemata should be inspected and be compared with the other schemata in the cluster. Then a percentage of weight should be considered for each schema in a way that the summation of the percentages for schemata in the cluster becomes 100. This way of calculating metrics is costlier in terms of time and expertise needed to judge the complexity of each schema.

All subjects of this experiment are extracted from one system which is implemented by one specific development team and one programming language. This issue should also be considered as a thread to validity of this study.

Chapter 5: Conclusions

5.1 Introduction

The goal of this chapter is to conclude the results of the master thesis which is aimed at examining *if Z-based specification measures can be used for predicting properties of related code implementations*.

For this reason three different studies are performed; a literature review in order to collect an appropriate set of metrics in Z specifications, another literature review in order to identify the code metrics which play a major role as input for outstanding software cost estimation models, and an experiment aimed at examining the correlation between collected specifications and code metrics. The next section will provide a summary for the results of each conducted study in this master thesis.

5.2 Study on Z Metrics

The literature review on the specification metrics is performed with focus on applicability on Z-specifications and availability of tools for measurement. This review resulted in three categories of metrics including total of eleven metrics, namely line of code in specifications, conceptual complexity, two metrics for logical (cyclomatic) complexity, definition use count, coupling, four metrics for cohesion, and overlap.

5.3 Study on Code Metrics

A literature review is performed in order to find the code metrics which can be used as input for outstanding software cost estimation models. Therefore, software cost estimation models are reviewed to find the outstanding ones and the useful code metrics as inputs parameters for them.

According to results of this study, COCOMO II is the most widely used among non-proprietary cost estimation models. The cost estimation model called Doty is also used despite of lack of enough precision. The code metric called Source Line of Code, or a more precise definition of that called Delivered Source Instructions, are used in both of these models, and a few other reviewed models, as input parameter for estimating the cost of software.

Being able to estimate complexity of code will also help to identify the risky parts of the implementation to apply special management and/or software development techniques as complexity is important in reducing the cost of software maintenance.

5.4 The Experiment

Because of lack of enough experimental subjects, an industrial project is broke down to smaller samples with a step-wise method which is explained in section 4-2-1 of chapter 4. Then, the measurement is performed on 28 extracted subjects each of which containing a set of Z schemata and a set of procedures/functions in code. List of study subjects is provided in Appendix A and the result of measurements in Appendix B. Full results of this study are available on internet[1].

Via a few statistical tests on measurement data, it revealed that metrics in Z-specifications are in correlation with size-based metrics in code. Nevertheless, because of validity threats which are explained in chapter 4, this master thesis is unable to make any claim about exact quality of correlation between Z metrics and code metrics. However, this study could not find any prove for correlation of specification metrics and the only studied metric for complexity of code, Cyclomatic Complexity.

5.5 Further Studies

Despite of proof for existence of correlation between specification and code metrics, no total cost estimation model is proposed in this thesis work. However, if the mentioned validity threats are removed or alleviated, then the last part of this study can be repeated, with more statistical tests, in order to investigate the exact quality of correlation between measures in specifications and measures in implementation. According to results of the literature review study for cost estimation models, a good estimation for size of code can lead to total software cost estimation model which is a mixing with existing cost estimation models like COCOMO II.

The further studies can be conducted once the software industry start to use Z specifications more widely and reveal the code of the software. It is a good situation to raise the need of an official repository of formal specifications and related codes to facilitate later studies on formal specifications. Moreover, in case of availability of more measurement tools for Z-specifications, the correlation of more metrics can be examined.

[1] http://goo.gl/yGnC7

Appendix A- List of Study Subjects

#	Procedures	Z Schemas
1	Admin.FinishOp	AdminFinishOp
2	Admin.Logon	AdminLogon
3	Admin.Logout	AdminLogout
4	Admin.StartOp	AdminStartOp
5	AdminToken.IsPresent Clock.TheCurrentTime Door.TheCurrentDoor Door.TheDoorAlarm Floppy.IsPresent Floppy.CurrentFloppy Latch.IsLocked Screen.SetMessage UserToken.IsPresent	DoorLatchAlarm UserToken AdminToken Finger Floppy Keyboard
6	Alarm.UpdateDevice Screen.UpdateScreen Updates.Activity Updates.EarlyActivity Latch.UpdateDevice Display.UpdateDevice Admin.SecurityOfficerIsPresent	TISEarlyUpdate TISUpdate
7	AuditLog.AddElementToLog AuditLog.TruncateLog	AddElementsToLog
8	AuditLog.ArchiveLog	ArchiveLog
9	AuditLog.ClearLogEntries	ClearLog
10	AuditLog.Init AdminToken.Init CertificateStore.Init ConfigData.Init Configuration.Init Display.Init Door.Init Enclave.Init Floppy.Init Keyboard.Init KeyStore.Init TISMain.Init UserToken.Init Latch.Init Admin.Init Stats.Init Screen.Init	StartContext StartNonEnrolledStation StartEnrolledStation TISStartup InitDoorLatchAlarm InitKeyStore InitConfig InitAdmin InitStats InitAuditLog InitIDStation

11	Cert.Attr.Auth.Construct Cert.Attr.Auth.TheRole Cert.Attr.Auth.TheClearance Cert.Attr.Auth.Extract Cert.Attr.IandA.TheTemplate IandACert.Extract PrivCert.TheRole PrivCert.TheClearance PrivCert.Extract AttrCert.TheBaseCert IDCert.TheSubject IDCert.ThePublicKey IDCert.Extract Cert.TheIssuer Cert.TheID Cert.TheMechanism Cert.GetData Cert.GetSignature CertProcessing.ExtractIDCertData CertProcessing.ExtractPrivCertData CertProcessing.ExtractIACertData CertProcessing.ExtractAuthCertData CertProcessing.ObtainRawData CertProcessing.ObtainSignatureData CertProcessing.ConstructAuthCert Cert.Attr.Auth.SetContents CertProcessing.AddAuthSignature UserToken.GetClass	NewAuthCert CertificateId Certificate IDCert CAIdCert AttCertificate PrivCert AuthCert IandACert
12	Cert.Attr.Auth.IsOK KeyStore.PrivateKeyPresent KeyStore.IssuerIsThisTIS Cert.IssuerKnown Cert.IsOK KeyStore.KeyMatchingIssuerPresent KeyStore.ThisTIS	CertIssuerKnown CertOK CertIssuerIsThisTIS AuthCertOK KeyStore
13	ConfigData.ValidateFile ConfigData.AuthPeriodIsEmpty ConfigData.GetAuthPeriod ConfigData.IsInEntryPeriod ConfigData.TheLatchUnlockDuration ConfigData.TheAlarmSilentDuration ConfigData.TheFingerWaitDuration ConfigData.TheTokenRemovalDuration ConfigData.TheEnclaveClearance ConfigData.TheSystemMaxFar ConfigData.TheAlarmThresholdEntries	Config
14	Display.SetValue	AuditDoor AuditLatch AuditAlarm AuditLogAlarm AuditDisplay AuditScreen NoChange LogChange
15	Enclave.CompleteFailedAdminLogon	FailedAdminTokenRemoved
16	Enclave.ResetScreenMessage	ResetScreenMessage UserEntryContext
17	UserToken.UpdateAuthCert	UpdateUserToken
18	Floppy.Write	UpdateFloppy
19	KeyStore.IsVerifiedBy KeyStore.Sign	KEYPART

20	Stats.AddFailedBio Stats.AddSuccessfulEntry Stats.AddFailedEntry Stats.AddSuccessfulBio	AddSuccessfulEntryToStats AddFailedEntryToStats AddSuccessfulBioCheckToStats AddFailedBioCheckToStats
21	ConfigData.TheDisplayFields Stats.DisplayStats	IDStation
22	TISMain.Processing TISMain.MainLoopBody	TISIdle TISAdminOp TISProcessing
23	UserEntry.FailedAccessTokenRemoved	FailedAccessTokenRemoved TISCompleteFailedAccess
24	UserEntry.Progress	TISUserEntryOp
25	UserEntry.ReadFinger UserEntry.UserTokenTorn UserEntry.ValidateUserToken UserEntry.ValidateFinger UserEntry.UpdateToken UserEntry.ValidateEntry UserEntry.StartEntry UserToken.GetIandATemplate UserToken.ReadAndCheck	ReadFingerOK NoFinger FingerTimeout TISReadFinger EntryOK WriteUserTokenFail WriteUserToken TISWriteUserToken WriteUserTokenOK ValidateFingerFail TISValidateFinger FingerOK ValidateFingerOK BioCheckRequired ValidateUserTokenOK BioCheckNotRequired ReadUserToken TISReadUserToken
26	UserEntry.UnlockDoor	UnlockDoorOK WaitingTokenRemoval TokenRemovalTimeout TISUnlockDoor
27	UserToken.AddAuthCert	AddAuthCertToUserToken

28	AdminToken.GetRole AdminToken.Interface.Poll AdminToken.Poll AdminToken.ReadAndCheck Bio.Poll Clock.Poll ConfigData.UpdateData ConfigData.WriteFile Configuration.UpdateData Display.ChangeDoorUnlockedMsg Door.LockDoor Door.Poll Door.UnlockDoor Door.UpdateDoorAlarm Enclave.AdminLogout Enclave.AdminOp Enclave.ArchiveLogOp Enclave.BadAdminTokenTear Enclave.CompleteFailedEnrolment Enclave.EnrolOp Enclave.OverrideDoorLockOp Enclave.ProgressAdminActivity Enclave.ReadEnrolmentData Enclave.ShutdownOp Enclave.StartAdminActivity Enclave.UpdateConfigDataOp Enclave.ValidateAdminToken Enclave.ValidateEnrolmentData Enrolment.Validate Floppy.CheckWrite Floppy.Read Keyboard.Interface.Poll Keyboard.Poll Keyboard.Read Latch.SetTimeout Latch.UpdateInternalLatch UserEntry.DisplayPollUpdate UserToken.Interface.Poll UserToken.Poll KeyStore.AddKey Poll.Activity	AdminTokenOK AdminTokenTimeout ClearLogThenAddElements CompleteFailedEnrolment EnrolContext FailedEnrolFloppyRemoved FinishArchiveLog FinishArchiveLogBadMatch FinishArchiveLogFail FinishArchiveLogNoFloppy FinishArchiveLogOK FinishUpdateConfigData FinishUpdateConfigDataFail FinishUpdateConfigDataOK LockDoor LoginAborted NoOpRequest OverrideDoorLockOK PollAdminToken PollDoor PollFinger PollFloppy PollKeyboard PollTime PollUserToken ReadAdminToken ReadEnrolmentData ReadEnrolmentFloppy RequestEnrolment ShutdownOK ShutdownWaitingDoor StartArchiveLog StartArchiveLogOK StartArchiveLogWaitingFloppy StartUpdateConfigData StartUpdateConfigOK StartUpdateConfigWaitingFloppy TISAdminLogon TISAdminLogout TISArchiveLogOp TISCompleteFailedAdminLogon TISCompleteTimeoutAdminLogout TISOverrideDoorLockOp TISPoll TISReadAdminToken TISShutdownOp TISStartAdminOp TISUpdateConfigDataOp TISValidateAdminToken TokenRemovedAdminLogout UnlockDoor ValidateAdminTokenFail ValidateAdminTokenOK ValidateEnrolmentData ValidateEnrolmentDataFail ValidateEnrolmentDataOK ValidateOpRequest ValidateOpRequestOK WaitingAdminTokenRemoval WaitingFloppyRemoval UpdateKeyStore

Appendix B- Measurement Results

Full results of this study are available on internet[1]

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Admin.FinishOp	5	1	4	1	AdminFinishOp	24.00	33.00	17.00	3457.00	114.00	0.67	0.67	0.67	0.67	1.00	0.24
Sample 01																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Admin.Logon	7	2	5	1	AdminLogon	31.00	36.00	16.00	3248.00	108.00	0.65	0.65	0.65	0.65	1.00	0.23
Sample 02																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Admin.Logout	5	1	4	1	AdminLogout	23	32	16	3248	111	0.65	0.65	0.65	0.65	1.00	0.24
Sample 03																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Admin.StartOp	6	1	5	1	AdminStartOp	31	39	19	3875	115	0.69	0.69	0.69	0.69	1.00	0.23
Sample 04																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AdminToken.IsPresent	5	1	4	1	AdminToken	4	2	1	1	0	0	0	0	0	0	0
Clock.TheCurrentTime	5	1	4	1	DoorLatchAlarm	15	16	1	1303	29	0.00	0.11	0.48	0.67	0.66	0.12
Door.TheCurrentDoor	5	1	4	1	Finger	4	2	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00
Door.TheDoorAlarm	5	1	4	1	Floppy	5	3	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00
Floppy.IsPresent	21	11	10	1	Keyboard	4	2	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00
Floppy.CurrentFloppy	5	1	4	1	UserToken	4	2	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00
Latch.IsLocked	5	1	4	1	**Summation**	**36**	**27**	**6**	**1308**	**29**	**0.00**	**0.11**	**0.48**	**0.67**	**0.66**	**0.12**
Screen.SetMessage	13	9	4	2	**Average**	**6**	**4.5**	**1**	**218**	**4.83**	**0.00**	**0.02**	**0.08**	**0.11**	**0.11**	**0.02**
UserToken.IsPresent	5	1	4	1												
Summation	**69**	**27**	**42**	**10**												
Average	**7.67**	**3.00**	**4.67**	**1.11**												
Sample 05																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Alarm.UpdateDevice	9	6	3	2	TISEarlyUpdate	98	78	11	2173	52	0.36	0.36	0.57	0.64	0.27	0.19
Screen.UpdateScreen	59	49	10	8	TISUpdate	208	180	43	9121	498	0.08	0.11	0.56	0.68	0.13	0.49
Updates.Activity	11	5	6	1	**Summation**	**306**	**258**	**54**	**11294**	**550**	**0.44**	**0.47**	**1.13**	**1.32**	**0.39**	**0.68**
Updates.EarlyActivity	6	2	4	1	**Average**	**153**	**129**	**27**	**5647**	**275**	**0.22**	**0.24**	**0.57**	**0.66**	**0.20**	**0.34**
Latch.UpdateDevice	17	13	4	3												
Display.UpdateDevice	32	25	7	4												
Admin.SecurityOfficerIsPresent	5	1	4	1												
Summation	**139**	**101**	**38**	**20**												
Average	**19.86**	**14.43**	**5.43**	**2.86**												
Sample 06																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AuditLog.AddElementToLog	23	14	9	2	AddElementsToLog	36	17	3	437	11	1	1	1	1	1	0.13
AuditLog.TruncateLog	17	11	6	1												
Summation	**40**	**25**	**15**	**3**												
Average	**20**	**12.5**	**7.5**	**1.5**												
Sample 07																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AuditLog.ArchiveLog	68	56	12	8	ArchiveLog	28	18	3	437	0	1	1	1	1	1	0.04
Sample 08																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AuditLog.ClearLogEntries	29	24	5	4	ClearLog	32	25	3	437	10	0.4	0.5	0.58	0.7	0.41	0.04
Sample 09																

[1] http://goo.gl/yGnC7

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AuditLog.Init	118	93	25	9	StartContext	187	173	51	8907	311	0.10	0.10	0.10	0.10	1.00	0.28
AdminToken.Init	5	2	3	1	StartNonEnrolledStation	202	181	53	8951	316	0.08	0.09	0.60	0.73	0.16	0.46
CertificateStore.Init	22	18	4	3	StartEnrolledStation	202	181	53	8951	316	0.08	0.09	0.60	0.73	0.16	0.46
ConfigData.Init	77	56	21	3	TISStartup	221	193	55	8995	321	0.07	0.08	0.63	0.72	0.11	0.44
Configuration.Init	4	1	3	1	InitDoorLatchAlarm	23	22	1	1391	29	0.77	0.77	0.77	0.77	1.00	0.09
Display.Init	13	9	4	2	InitKeyStore	12	10	1	439	3	0.80	0.80	0.80	0.80	1.00	0.04
Door.Init	6	3	3	1	InitConfig	23	18	1	437	0	0.00	0.13	0.17	0.25	0.00	0.02
Enclave.Init	8	5	3	2	InitAdmin	21	31	1	3083	56	0.64	0.64	0.64	0.64	0.99	0.23
Floppy.Init	47	34	13	3	InitStats	14	11	1	89	0	0.00	0.25	0.25	0.25	0.96	0.00
Keyboard.Init	4	1	3	1	InitAuditLog	10	7	1	1	0	0.00	0.50	0.50	0.50	0.00	0.00
KeyStore.Init	35	23	12	4	InitIDStation	187	170	1	8907	310	0.00	0.01	0.18	0.54	0.00	0.35
TISMain.Init	33	30	3	2	**Summation**	**1102**	**997**	**219**	**50151**	**1662**	**2.53**	**3.46**	**5.24**	**6.03**	**5.38**	**2.36**
UserToken.Init	5	2	3	1	**Average**	**100.18**	**90.64**	**19.91**	**4559.18**	**151.09**	**0.23**	**0.31**	**0.48**	**0.55**	**0.49**	**0.21**
Latch.Init	5	2	3	1												
Admin.Init	6	2	4	1												
Stats.Init	8	4	4	1												
Screen.Init	35	28	7	3												
Summation	**431**	**313**	**118**	**39**												
Average	**25.35**	**18.41**	**6.94**	**2.29**												

Sample 10

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Cert.Attr.Auth.Construct	42	26	16	1	NewAuthCert	54	42	1	875	3	0	0.12	0.76	0.94	0.72	0.07
Cert.Attr.Auth.TheRole	5	1	4	1	CertificateId	3	1	1	1	0	0	0	0	0	0	0
Cert.Attr.Auth.TheClearance	5	1	4	1	Certificate	5	3	1	1	0	0	0	0	0	0	0
Cert.Attr.Auth.Extract	37	27	10	1	IDCert	5	3	1	1	0	0	0	0	0	0	0
Cert.Attr.IandA.TheTemplate	5	1	4	1	CAIdCert	15	11	1	1	0	1	1	1	1	1	0
Cert.Attr.IandA.Extract	36	26	10	1	AttCertificate	5	3	1	1	0	0	0	0	0	0	0
UserToken.GetClass	5	1	4	1	PrivCert	5	3	1	1	0	0	0	0	0	0	0
Cert.Attr.Priv.TheRole	5	1	4	1	AuthCert	5	3	1	1	0	0	0	0	0	0	0
Cert.Attr.Priv.TheClearance	5	1	4	1	IandACert	4	2	1	1	0	0	0	0	0	0	0
Cert.Attr.Priv.Extract	37	27	10	1	**Summation**	**101**	**71**	**9**	**883**	**3**	**1.00**	**1.12**	**1.76**	**1.94**	**1.72**	**0.07**
CertProcessing.AddAuthSignature	10	3	7	1	**Average**	**11.22**	**7.89**	**1.00**	**98.11**	**0.33**	**0.11**	**0.12**	**0.20**	**0.22**	**0.19**	**0.01**
Cert.Attr.TheBaseCert	5	1	4	1												
Cert.Attr.Auth.SetContents	18	7	11	1												
Cert.ID.TheSubject	5	1	4	1												
Cert.ID.ThePublicKey	5	1	4	1												
Cert.ID.Extract	53	42	11	2												
CertProcessing.ConstructAuthCert	31	2	29	1												
Cert.TheIssuer	5	1	4	1												
Cert.TheID	5	1	4	1												
Cert.TheMechanism	6	1	5	1												
Cert.GetData	11	4	7	1												
Cert.GetSignature	11	4	7	1												
CertProcessing.ExtractIDCertData	33	24	9	1												
CertProcessing.ExtractPrivCertData	32	23	9	1												
CertProcessing.ExtractIACertData	30	21	9	1												
CertProcessing.ExtractAuthCertData	32	23	9	1												
CertProcessing.ObtainRawData	13	5	8	1												
CertProcessing.ObtainSignatureData	13	5	8	1												
Summation	**500**	**281**	**219**	**29**												
Average	**17.86**	**40.14**	**31.29**	**4.14**												

Sample 11

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Cert.Attr.Auth.IsOK	11	5	6	1	CertIssuerKnown	17	14	1	439	3	0.75	0.75	0.75	0.75	0.98	0.04
KeyStore.PrivateKeyPresent	5	1	4	1	CertOK	22	17	1	439	3	0.80	0.80	0.80	0.80	0.98	0.04
KeyStore.IssuerIsThisTIS	12	6	6	2	CertIssuerIsThisTIS	18	15	1	439	3	0.80	0.80	0.80	0.80	0.76	0.04
Cert.IssuerKnown	8	3	5	1	AuthCertOK	32	25	1	439	3	0.86	0.86	0.86	0.86	0.69	0.04
Cert.IsOK	19	12	7	2	KeyStore	6	6	1	439	3	0.67	0.67	0.67	0.67	1.00	0.04
KeyStore.KeyMatchingIssuerPresent	9	3	6	1	**Summation**	**95**	**77**	**5**	**2195**	**15**	**3.87**	**3.87**	**3.87**	**3.87**	**4.41**	**0.21**
KeyStore.ThisTIS	5	1	4	1	**Average**	**19.00**	**15.40**	**1.00**	**439.00**	**3.00**	**0.77**	**0.77**	**0.77**	**0.77**	**0.88**	**0.04**
Summation	**69**	**31**	**38**	**9**												
Average	**9.86**	**4.43**	**5.43**	**1.29**												

Sample 12

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
ConfigData.ValidateFile	402	273	129	15	Config	12	10	1	437	0	1	1	1	1	1	0.04
ConfigData.AuthPeriodIsEmpty	19	14	5	4												
ConfigData.GetAuthPeriod	19	12	7	3												
ConfigData.IsInEntryPeriod	5	1	4	1												
ConfigData.TheLatchUnlockDuration	5	1	4	1												
ConfigData.TheAlarmSilentDuration	5	1	4	1												
ConfigData.TheFingerWaitDuration	5	1	4	1												
ConfigData.TheTokenRemovalDuration	5	1	4	1												
ConfigData.TheEnclaveClearance	5	1	4	1												
ConfigData.TheSystemMaxFar	5	1	4	1												
ConfigData.TheAlarmThresholdEntries	15	10	5	2												
Summation	**490**	**316**	**174**	**31**												
Average	**44.55**	**28.73**	**15.82**	**2.82**												

Sample 13

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Display.SetValue	13	9	4	2	AuditDoor	57	37	9	1739	49	0.77	0.77	0.77	0.77	1.00	0.17
					AuditLatch	57	37	9	1739	46	0.77	0.77	0.77	0.77	1.00	0.17
					AuditAlarm	58	37	9	1739	43	0.77	0.77	0.77	0.77	1.00	0.17
					AuditLogAlarm	41	20	3	437	14	1.00	1.00	1.00	1.00	0.97	0.12
					AuditDisplay	152	134	41	8687	377	0.62	0.62	0.64	0.67	0.54	0.49
					AuditScreen	152	134	41	8687	330	0.62	0.62	0.64	0.67	0.52	0.49
					NoChange	138	135	42	8896	398	0.65	0.66	0.68	0.69	0.47	0.47
					LogChange	199	171	42	8896	481	0.43	0.57	0.64	0.71	0.29	0.46
					Summation	**854**	**705**	**196**	**40820**	**1738**	**5.63**	**5.77**	**5.91**	**6.04**	**5.79**	**2.55**
					Average	**106.75**	**88.13**	**24.50**	**5102.50**	**217.25**	**0.70**	**0.72**	**0.74**	**0.76**	**0.72**	**0.32**

Sample 14

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Enclave.CompleteFailedAdminLogon	12	9	3	1	FailedAdminTokenRemoved	209	175	43	9105	557	0.08	0.10	0.59	0.70	0.13	0.49
Sample 15																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Enclave.ResetScreenMessage	13	9	4	4	ResetScreenMessage	39	56	15	3039	370	0.48	0.48	0.60	0.61	0.13	0.28
					UserEntryContext	186	185	42	8896	677	0.00	0.08	0.57	0.66	0.00	0.50
					Summation	**225**	**241**	**57**	**11935**	**1047**	**0.48**	**0.56**	**1.17**	**1.27**	**0.13**	**0.78**
					Average	**112.50**	**120.50**	**28.50**	**5967.50**	**523.50**	**0.24**	**0.28**	**0.59**	**0.63**	**0.06**	**0.39**
Sample 16																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserToken.UpdateAuthCert	23	15	8	2	UpdateUserToken	156	149	41	8687	312	0.11	0.11	0.39	0.67	0.51	0.50
Sample 17																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Floppy.Write	46	37	9	4	UpdateFloppy	173	164	41	8687	394	0.09	0.10	0.53	0.61	0.21	0.48
Sample 18																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
KeyStore.IsVerifiedBy	36	24	12	3	KEYPART	3	1	1	1	0	0	0	0	0	0	0
KeyStore.Sign	37	26	11	3												
Summation	**73**	**50**	**23**	**6**												
Average	**36.5**	**25**	**11.5**	**3**												
Sample 19																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Stats.AddFailedBio	7	3	4	2	AddSuccessfulEntryToStats	14	11	1	1	24	0	0.25	0.25	0.25	0	0.00
Stats.AddSuccessfulEntry	7	3	4	2	AddFailedEntryToStats	14	11	1	1	24	0	0.25	0.25	0.25	0	0.01
Stats.AddFailedEntry	7	3	4	2	AddSuccessfulBioCheckToStats	14	11	1	1	24	0	0.25	0.25	0.25	0	0.00
Stats.AddSuccessfulBio	7	3	4	2	AddFailedBioCheckToStats	14	11	1	1	24	0	0.25	0.25	0.25	0	0.00
Summation	**28**	**12**	**16**	**8**	**Summation**	**56**	**44**	**4**	**4**	**96**	**0**	**1**	**1**	**1**	**0**	**0.02**
Average	**7**	**3**	**4**	**2**	**Average**	**14**	**11**	**1**	**1**	**24**	**0**	**0.25**	**0.25**	**0.25**	**0**	**0.00**
Sample 20																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
ConfigData.TheDisplayFields	30	13	17	1	IDStation	126	126	1	8687	310	0	0.02	0.55	0.66	0.54	0.49
Stats.DisplayStats	12	4	8	1												
Summation	**42**	**17**	**25**	**2**												
Average	**21**	**8.5**	**12.5**	**1**												
Sample 21																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
TISMain.Processing	28	21	7	6	TISIdle	142	139	1	9732	371	0	0.02	0.59	0.68	0.53	0.48
TISMain.MainLoopBody	13	10	3	3	TISAdminOp	540	445	80	16838	1771	0	0.04	0.65	0.76	0.00	0.36
Summation	**41**	**31**	**10**	**9**	TISProcessing	1369	1049	184	38574	3300	0	0.01	0.55	0.81	0.00	0.23
Average	**20.5**	**15.5**	**5**	**4.5**	**Summation**	**2051**	**1633**	**265**	**65144**	**5442**	**0**	**0.06**	**1.79**	**2.25**	**0.53**	**1.07**
					Average	**683.67**	**544.33**	**88.33**	**21714.67**	**1814.00**	**0.00**	**0.02**	**0.60**	**0.75**	**0.18**	**0.36**
Sample 22																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserEntry.FailedAccessTokenRemoved	14	10	4	1	FailedAccessTokenRemoved	229	205	44	9314	791	0	0.01	0.50	0.69	0.00	0.49
					TISCompleteFailedAccess	244	221	48	10150	847	0	0.01	0.49	0.68	0.00	0.47
					Summation	**473**	**426**	**92**	**19464**	**1638**	**0**	**0.02**	**0.99**	**1.37**	**0.00**	**0.96**
					Average	**236.50**	**213.00**	**46.00**	**9732.00**	**819.00**	**0**	**0.01**	**0.49**	**0.68**	**0.00**	**0.48**
Sample 23																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserEntry.Progress	24	17	7	7	TISUserEntryOp	615	497	92	19346	1862	0	0.02	0.45	0.76	0.00	0.33
Sample 24																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserEntry.ReadFinger	33	28	5	4	ReadFingerOK	222	200	45	9523	768	0.00	0.08	0.60	0.68	0.00	0.49
UserEntry.UserTokenTorn	14	10	4	1	NoFinger	158	151	44	9314	341	0.11	0.11	0.11	0.11	1.00	0.29
UserEntry.ValidateUserToken	67	60	7	4	FingerTimeout	221	199	44	9314	767	0.00	0.08	0.59	0.67	0.00	0.50
UserEntry.ValidateFinger	62	43	19	4	TISReadFinger	276	242	52	10986	969	0.00	0.01	0.52	0.71	0.00	0.46
UserEntry.UpdateToken	32	27	5	4	EntryOK	224	200	44	9314	774	0.00	0.08	0.60	0.68	0.00	0.49
UserEntry.ValidateEntry	32	28	4	3	WriteUserTokenFail	241	208	46	9732	789	0.00	0.07	0.58	0.69	0.00	0.49
UserEntry.StartEntry	5	2	3	1	WriteUserToken	259	222	48	10150	865	0.00	0.07	0.59	0.69	0.00	0.48
UserToken.GetIandATemplate	5	1	4	1	TISWriteUserToken	288	246	50	10568	963	0.00	0.01	0.51	0.70	0.00	0.46
UserToken.ReadAndCheck	161	119	42	2	WriteUserTokenOK	241	208	46	9732	789	0.00	0.07	0.58	0.69	0.00	0.49
Summation	**411**	**318**	**93**	**24**	ValidateFingerFail	229	205	44	9314	791	0.00	0.01	0.50	0.69	0.00	0.49
Average	**45.67**	**35.33**	**10.33**	**2.67**	TISValidateFinger	282	245	48	10150	967	0.00	0.03	0.42	0.72	0.00	0.45
					FingerOK	14	10	1	1	0	1.00	1.00	1.00	1.00	1.00	0.00
					ValidateFingerOK	231	205	44	9314	791	0.00	0.01	0.50	0.69	0.00	0.49
					BioCheckRequired	240	207	47	9941	772	0.00	0.08	0.59	0.68	0.00	0.49
					ValidateUserTokenOK	260	221	49	10359	849	0.00	0.08	0.60	0.69	0.00	0.48
					BioCheckNotRequired	223	199	44	9314	767	0.00	0.08	0.59	0.67	0.00	0.50
					ReadUserToken	224	200	45	9523	787	0.00	0.08	0.60	0.68	0.00	0.49
					TISReadUserToken	227	202	45	9523	787	0.00	0.08	0.60	0.68	0.00	0.49
					Summation	**4060**	**3570**	**786**	**166072**	**13536**	**1.11**	**2.01**	**10.07**	**12.11**	**2.00**	**8.02**
					Average	**225.56**	**198.33**	**43.67**	**9226.22**	**752.00**	**0.06**	**0.11**	**0.56**	**0.67**	**0.11**	**0.45**
Sample 25																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserEntry.UnlockDoor	24	20	4	3	UnlockDoorOK	238	212	44	9314	815	0.00	0.01	0.53	0.70	0.00	0.48
					WaitingTokenRemoval	159	153	45	9523	374	0.10	0.10	0.10	0.10	1.00	0.28
					TokenRemovalTimeout	222	200	45	9523	773	0.00	0.08	0.60	0.68	0.00	0.49
					TISUnlockDoor	271	240	51	10777	950	0.00	0.01	0.54	0.71	0.00	0.45
					Summation	**890**	**805**	**185**	**39137**	**2912**	**0.10**	**0.20**	**1.77**	**2.19**	**1.00**	**1.71**
					Average	**222.50**	**201.25**	**46.25**	**9784.25**	**728.00**	**0.03**	**0.05**	**0.44**	**0.55**	**0.25**	**0.43**
Sample 26																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
UserToken.AddAuthCert	51	41	10	4	AddAuthCertToUserToken	42	28	7	1293	26	0.89	0.89	0.89	0.89	0.51	0.08
Sample 27																

Procs	CLC	LE	LD	CC	Z Schemata	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
AdminToken.GetRole	5	1	4	1	AdminTokenOK	25	15	1	439	3	0.00	0.40	0.40	0.40	0.00	0.04
AdminToken.Interface.Poll	4	1	3	1	AdminTokenTimeout	235	206	46	9732	826	0.00	0.07	0.59	0.69	0.00	0.49
AdminToken.Poll	5	2	3	1	ClearLogThenAddElements	35	28	3	437	10	0.36	0.45	0.53	0.64	0.41	0.04
AdminToken.ReadAndCheck	136	103	33	4	CompleteFailedEnrolment	196	178	45	9523	487	0.08	0.10	0.56	0.69	0.16	0.48
Bio.Poll	5	1	4	1	EnrolContext	170	160	41	8687	318	0.10	0.11	0.38	0.65	0.40	0.50
Clock.Poll	4	1	3	1	FailedEnrolFloppyRemoved	184	169	43	9105	460	0.09	0.10	0.57	0.70	0.16	0.49
ConfigData.UpdateData	30	13	17	1	FinishArchiveLog	207	181	49	10359	561	0.08	0.09	0.60	0.73	0.12	0.48
ConfigData.WriteFile	150	123	27	3	FinishArchiveLogBadMatch	225	185	48	10150	570	0.08	0.09	0.61	0.73	0.11	0.48
Configuration.UpdateData	112	74	38	3	FinishArchiveLogFail	241	198	50	10568	579	0.07	0.09	0.62	0.73	0.10	0.47
Display.ChangeDoorUnlockedMsg	7	3	4	2	FinishArchiveLogNoFloppy	223	184	47	9941	565	0.08	0.09	0.61	0.72	0.11	0.49
Door.LockDoor	7	4	3	1	FinishArchiveLogOK	207	181	49	10359	561	0.08	0.09	0.60	0.73	0.12	0.48
Door.Poll	34	28	6	4	FinishUpdateConfigData	225	184	48	10150	565	0.08	0.09	0.61	0.73	0.11	0.48
Door.UnlockDoor	15	11	4	1	FinishUpdateConfigDataFail	224	184	47	9941	565	0.08	0.09	0.61	0.72	0.11	0.49
Door.UpdateDoorAlarm	27	21	6	3	FinishUpdateConfigDataOK	225	184	48	10150	565	0.08	0.09	0.61	0.73	0.11	0.48
Enclave.AdminLogout	27	23	4	4	LockDoor	24	23	7	1303	56	0.79	0.79	0.79	0.79	1.00	0.12
Enclave.AdminOp	14	10	4	4	LoginAborted	230	202	44	9314	799	0.00	0.08	0.59	0.68	0.00	0.50
Enclave.ArchiveLogOp	56	42	14	2	NoOpRequest	185	170	46	9732	492	0.09	0.10	0.51	0.70	0.23	0.49
Enclave.BadAdminTokenTear	11	8	3	1	OverrideDoorLockOK	230	190	46	9732	549	0.08	0.09	0.65	0.73	0.08	0.48
Enclave.CompleteFailedEnrolment	8	5	3	2	PollAdminToken	29	25	2	210	13	0.75	0.75	0.75	0.75	0.55	0.01
Enclave.EnrolOp	13	9	4	3	PollDoor	40	37	7	1303	35	0.75	0.75	0.75	0.75	1.00	0.12
Enclave.OverrideDoorLockOp	15	11	4	1	PollFinger	28	25	2	210	2	0.75	0.75	0.75	0.75	0.55	0.01
Enclave.ProgressAdminActivity	16	11	5	4	PollFloppy	30	27	2	210	23	0.40	0.60	0.67	0.80	0.11	0.01
Enclave.ReadEnrolmentData	12	9	3	2	PollKeyboard	29	25	3	419	3	0.75	0.75	0.75	0.75	1.00	0.01
Enclave.ShutdownOp	21	17	4	2	PollTime	39	35	7	1303	39	0.70	0.70	0.70	0.70	1.00	0.12
Enclave.StartAdminActivity	56	37	19	3	PollUserToken	28	25	2	210	27	0.75	0.75	0.75	0.75	0.55	0.01
Enclave.UpdateConfigDataOp	27	21	6	4	ReadAdminToken	209	176	45	9523	543	0.08	0.10	0.57	0.71	0.15	0.49
Enclave.ValidateAdminToken	39	33	6	3	ReadEnrolmentData	201	182	45	9523	506	0.08	0.09	0.59	0.70	0.13	0.47
Enclave.ValidateEnrolmentData	31	25	6	2	ReadEnrolmentFloppy	184	169	43	9105	460	0.09	0.10	0.57	0.70	0.16	0.49
Enrolment.Validate	119	85	34	7	RequestEnrolment	184	169	43	9105	365	0.09	0.10	0.51	0.69	0.23	0.49
Floppy.CheckWrite	22	17	5	3	ShutdownOK	223	186	45	9523	489	0.08	0.09	0.64	0.73	0.08	0.48
Floppy.Read	32	23	9	3	ShutdownWaitingDoor	184	169	44	9314	463	0.09	0.10	0.55	0.70	0.19	0.49
Keyboard.Interface.Poll	4	1	3	1	StartArchiveLog	297	251	51	10777	1139	0.00	0.06	0.61	0.72	0.00	0.46
Keyboard.Poll	4	1	3	1	StartArchiveLogOK	195	175	45	9523	526	0.08	0.09	0.61	0.72	0.13	0.48
Keyboard.Read	12	5	7	1	StartArchiveLogWaitingFloppy	191	173	45	9523	525	0.08	0.10	0.58	0.70	0.16	0.49
KeyStore.AddKey	31	22	9	3	StartUpdateConfigData	292	248	51	10777	1116	0.00	0.06	0.62	0.72	0.00	0.47
Latch.SetTimeout	5	1	4	1	StartUpdateConfigOK	191	173	45	9523	502	0.08	0.10	0.58	0.70	0.16	0.49
Latch.UpdateInternalLatch	22	17	5	3	StartUpdateConfigWaitingFloppy	191	173	45	9523	525	0.08	0.10	0.58	0.70	0.16	0.49
Poll.Activity	10	6	4	1	TISAdminLogon	343	274	58	12240	1250	0.00	0.06	0.63	0.74	0.00	0.44
UserEntry.DisplayPollUpdate	12	8	4	3	TISAdminLogout	295	249	53	11195	1111	0.00	0.06	0.61	0.72	0.00	0.46
UserToken.Interface.Poll	4	1	3	1	TISArchiveLogOp	376	312	64	13494	1353	0.00	0.05	0.63	0.74	0.00	0.43
UserToken.Poll	5	2	3	1	TISCompleteFailedAdminLogon	221	184	45	9523	582	0.08	0.09	0.58	0.70	0.13	0.48
Summation	**1169**	**836**	**333**	**93**	TISCompleteTimeoutAdminLogout	224	186	45	9523	582	0.08	0.09	0.58	0.70	0.13	0.48
Average	**28.51**	**20.39**	**8.12**	**2.27**	TISOverrideDoorLockOp	294	249	50	10568	1036	0.00	0.06	0.64	0.72	0.00	0.47
					TISPoll	301	266	52	10986	615	0.67	0.68	0.70	0.71	0.27	0.36
					TISReadAdminToken	213	178	45	9523	543	0.08	0.10	0.57	0.71	0.15	0.49
					TISShutdownOp	243	202	48	10150	628	0.07	0.08	0.65	0.74	0.07	0.46
					TISStartAdminOp	252	206	54	11404	650	0.07	0.08	0.63	0.74	0.09	0.45
					TISUpdateConfigDataOp	355	294	60	12658	1319	0.00	0.05	0.64	0.73	0.00	0.44
					TISValidateAdminToken	292	239	50	10568	1056	0.00	0.07	0.62	0.72	0.00	0.47
					TokenRemovedAdminLogout	234	205	45	9523	826	0.00	0.07	0.60	0.69	0.00	0.50
					UnlockDoor	36	33	9	1739	53	0.80	0.80	0.80	0.80	1.00	0.15
					UpdateKeyStore	33	19	6	445	4	0.88	0.88	0.88	0.88	0.77	0.04
					ValidateAdminTokenFail	209	175	44	9314	506	0.08	0.10	0.57	0.70	0.14	0.49
					ValidateAdminTokenOK	221	180	44	9314	553	0.08	0.10	0.60	0.71	0.11	0.49
					ValidateEnrolmentData	237	192	46	9732	582	0.08	0.09	0.62	0.72	0.10	0.47
					ValidateEnrolmentDataFail	207	174	43	9105	468	0.09	0.10	0.59	0.70	0.13	0.49
					ValidateEnrolmentDataOK	216	178	44	9314	446	0.08	0.10	0.59	0.70	0.14	0.49
					ValidateOpRequest	249	204	54	11404	650	0.07	0.08	0.63	0.74	0.09	0.45
					ValidateOpRequestOK	224	187	51	10777	592	0.08	0.09	0.61	0.74	0.11	0.47
					WaitingAdminTokenRemoval	169	158	43	9105	344	0.09	0.11	0.37	0.63	0.40	0.50
					WaitingFloppyRemoval	169	158	43	9105	346	0.09	0.11	0.37	0.63	0.40	0.50
					Summation	**11669**	**9967**	**2376**	**499935**	**31927**	**11.36**	**13.27**	**37.29**	**43.53**	**13.83**	**23.99**
					Average	**191.30**	**163.39**	**38.95**	**8195.66**	**523.39**	**0.19**	**0.22**	**0.61**	**0.71**	**0.23**	**0.39**
Sample 28																

Sample#	CLC	LE	LD	CC	LOC	CC	CyclL	CyclU	DU	Tightness	MinCov	Cov	MaxCov	Overlap	Coupling
Sample01	5	1	4	1	24	33	17	3457	114	0.67	0.67	0.67	0.67	1.00	0.24
Sample02	7	2	5	1	31	36	16	3248	108	0.65	0.65	0.65	0.65	1.00	0.23
Sample03	5	1	4	1	23	32	16	3248	111	0.65	0.65	0.65	0.65	1.00	0.24
Sample04	6	1	5	1	31	39	19	3875	115	0.69	0.69	0.69	0.69	1.00	0.23
Sample05	69	27	42	10	36	27	6	1308	29	0.00	0.02	0.08	0.11	0.11	0.02
Sample06	139	101	38	20	306	258	54	11294	550	0.22	0.24	0.57	0.66	0.20	0.34
Sample07	40	25	15	3	36	17	3	437	11	1.00	1.00	1.00	1.00	1.00	0.13
Sample08	68	56	12	8	28	18	3	437	0	1.00	1.00	1.00	1.00	1.00	0.04
Sample09	29	24	5	4	32	25	3	437	10	0.40	0.50	0.58	0.70	0.41	0.04
Sample10	431	313	118	39	1102	997	219	50151	1662	0.23	0.31	0.48	0.55	0.49	0.21
Sample11	500	281	219	29	101	71	9	883	3	0.11	0.12	0.20	0.22	0.19	0.01
Sample12	69	31	38	9	95	77	5	2195	15	0.77	0.77	0.77	0.77	0.88	0.04
Sample13	490	316	174	31	12	10	1	437	0	1.00	1.00	1.00	1.00	1.00	0.04
Sample14	13	9	4	2	854	705	196	40820	1738	0.70	0.72	0.74	0.76	0.72	0.32
Sample15	12	9	3	1	209	175	43	9105	557	0.08	0.10	0.59	0.70	0.13	0.49
Sample16	13	9	4	4	225	241	57	11935	1047	0.24	0.28	0.59	0.63	0.06	0.39
Sample17	23	15	8	2	156	149	41	8687	312	0.11	0.11	0.39	0.67	0.51	0.50
Sample18	46	37	9	4	173	164	41	8687	394	0.09	0.10	0.53	0.61	0.21	0.48
Sample19	73	50	23	6	3	1	1	1	0	0.00	0.00	0.00	0.00	0.00	0.00
Sample20	28	12	16	8	56	44	4	4	96	0.00	0.25	0.25	0.25	0.00	0.00
Sample21	42	17	25	2	126	126	1	8687	310	0.00	0.02	0.55	0.66	0.54	0.49
Sample22	41	31	10	9	2051	1633	265	65144	5442	0.00	0.02	0.60	0.75	0.18	0.36
Sample23	14	10	4	1	473	426	92	19464	1638	0.00	0.01	0.49	0.68	0.00	0.48
Sample24	24	17	7	7	615	497	92	19346	1862	0.00	0.02	0.45	0.76	0.00	0.33
Sample25	411	318	93	24	4060	3570	786	166072	13536	0.06	0.11	0.56	0.67	0.11	0.45
Sample26	24	20	4	3	890	805	185	39137	2912	0.03	0.05	0.44	0.55	0.25	0.43
Sample27	51	41	10	4	42	28	7	1293	26	0.89	0.89	0.89	0.89	0.51	0.08
Sample28	1169	836	333	93	11669	9967	2376	499935	31927	0.19	0.22	0.61	0.71	0.23	0.39

Summary of measurement for all samples

Appendix C- Analysis Results

Table C-1- Regression analysis of **Count Line Code**

Regression Statistics	
Multiple R	0.92
R Square	0.85
Adjusted R Square	0.75
Standard Error	126.11
Observations	28

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	11	1482396.145	134763.2859	8.4741271	9.19942E-05
Residual	16	254446.5689	15902.91056		
Total	27	1736842.714			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	120.21	91.61	1.31	0.21	-74.00	314.42	-74.00	314.42
LOC	-1.79	0.75	-2.39	0.03	-3.37	-0.20	-3.37	-0.20
CC	3.26	1.31	2.48	0.02	0.48	6.05	0.48	6.05
CyclL	4.89	4.16	1.17	0.26	-3.94	13.71	-3.94	13.71
CyclU	-0.03	0.03	-1.24	0.23	-0.09	0.02	-0.09	0.02
DU	-0.18	0.08	-2.28	0.04	-0.34	-0.01	-0.34	-0.01
Tightness	566.57	609.47	0.93	0.37	-725.46	1858.59	-725.46	1858.59
MinCov	-1473.22	686.11	-2.15	0.05	-2927.72	-18.72	-2927.72	-18.72
Cov	1076.74	802.23	1.34	0.20	-623.91	2777.39	-623.91	2777.39
MaxCov	-261.73	494.51	-0.53	0.60	-1310.03	786.58	-1310.03	786.58
Overlap	194.55	203.04	0.96	0.35	-235.87	624.97	-235.87	624.97
Coupling	-1004.59	391.26	-2.57	0.02	-1834.02	-175.17	-1834.02	-175.17

Table C-2- Regression analysis of **Lines Executable**

Regression Statistics	
Multiple R	0.94
R Square	0.88
Adjusted R Squ	0.81
Standard Error	78.80
Observations	28

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	11	760576.8997	69143.35451	11.1342	0.00002
Residual	16	99359.95748	6209.997343		
Total	27	859936.8571			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	60.74	57.25	1.06	0.30	-60.62	182.10	-60.62	182.10
LOC	-1.35	0.47	-2.89	0.01	-2.34	-0.36	-2.34	-0.36
CC	2.48	0.82	3.02	0.01	0.74	4.22	0.74	4.22
CyclL	3.14	2.60	1.21	0.24	-2.37	8.66	-2.37	8.66
CyclU	-0.02	0.02	-1.39	0.18	-0.06	0.01	-0.06	0.01
DU	-0.13	0.05	-2.65	0.02	-0.23	-0.03	-0.23	-0.03
Tightness	377.12	380.86	0.99	0.34	-430.27	1184.50	-430.27	1184.50
MinCov	-910.95	428.75	-2.12	0.05	-1819.86	-2.04	-1819.86	-2.04
Cov	669.17	501.31	1.33	0.20	-393.56	1731.91	-393.56	1731.91
MaxCov	-152.67	309.02	-0.49	0.63	-807.75	502.42	-807.75	502.42
Overlap	99.66	126.88	0.79	0.44	-169.31	368.62	-169.31	368.62
Coupling	-605.06	244.49	-2.47	0.02	-1123.36	-86.76	-1123.36	-86.76

Table C-3- Regression analysis of **Lines Declarative**

Regression Statistics	
Multiple R	0.85
R Square	0.73
Adjusted R Squ	0.54
Standard Error	54.43
Observations	28

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	11	125716.828	11428.80255	3.8575103	0.007378746
Residual	16	47403.85055	2962.74066		
Total	27	173120.6786			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	56.01	39.54	1.42	0.18	-27.82	139.83	-27.82	139.83
LOC	-0.44	0.32	-1.36	0.19	-1.12	0.25	-1.12	0.25
CC	0.97	0.57	1.71	0.11	-0.23	2.17	-0.23	2.17
CyclL	2.81	1.80	1.57	0.14	-0.99	6.62	-0.99	6.62
CyclU	-0.02	0.01	-1.51	0.15	-0.04	0.01	-0.04	0.01
DU	-0.07	0.03	-2.15	0.05	-0.14	0.00	-0.14	0.00
Tightness	231.53	263.07	0.88	0.39	-326.15	789.20	-326.15	789.20
MinCov	-673.96	296.15	-2.28	0.04	-1301.76	-46.16	-1301.76	-46.16
Cov	579.60	346.26	1.67	0.11	-154.45	1313.64	-154.45	1313.64
MaxCov	-205.97	213.44	-0.96	0.35	-658.45	246.51	-658.45	246.51
Overlap	94.01	87.64	1.07	0.30	-91.77	279.79	-91.77	279.79
Coupling	-389.66	168.88	-2.31	0.03	-747.66	-31.66	-747.66	-31.66

Table C-4- Regression analysis of **Cyclomatic Complexity**

Regression Statistics	
Multiple R	0.67
R Square	0.44
Adjusted R Squ	0.06
Standard Error	29.65
Observations	28

ANOVA

	df	*SS*	*MS*	*F*	*Significance F*
Regression	11	11274.475	1024.952	1.166	0.379
Residual	16	14065.382	879.086		
Total	27	25339.857			

	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	5.78	21.54	0.27	0.79	-39.88	51.44	-39.88	51.44
LOC	-0.12	0.18	-0.65	0.52	-0.49	0.26	-0.49	0.26
CC	0.40	0.31	1.30	0.21	-0.25	1.06	-0.25	1.06
CyclL	1.27	0.98	1.30	0.21	-0.80	3.35	-0.80	3.35
CyclU	-0.01	0.01	-1.42	0.18	-0.02	0.00	-0.02	0.00
DU	-0.04	0.02	-2.04	0.06	-0.08	0.00	-0.08	0.00
Tightness	60.40	143.30	0.42	0.68	-243.37	364.18	-243.37	364.18
MinCov	-180.65	161.31	-1.12	0.28	-522.62	161.33	-522.62	161.33
Cov	218.94	188.62	1.16	0.26	-180.91	618.78	-180.91	618.78
MaxCov	-98.11	116.27	-0.84	0.41	-344.58	148.37	-344.58	148.37
Overlap	10.30	47.74	0.22	0.83	-90.90	111.50	-90.90	111.50
Coupling	-58.85	91.99	-0.64	0.53	-253.85	136.16	-253.85	136.16

References

[1] Bowen, J.P.; Hinchey, M.G., "Ten commandments of formal methods," Computer , vol.28, no.4, pp.56-63, Apr 1995.

[2] Jonathan P. Bowen, Michael G. Hinchey, "Seven More Myths of Formal Methods," IEEE Software, vol. 12, no. 4, pp. 34-41, July 1995.

[3] Khosrow-Pour M., Dictionary of information science and technology, IDEA GROUP REFERENCE, 2007.

[4] Bowen, J.P.: Formal Specification and Documentation Using Z: A Case Study Approach. Int. Thomson Computer Press, 1996.

[5] Spivey, J.M., "An introduction to Z and formal specifications", Software engineering journal, 1989.

[6] Perfect Developer: A tool for Object-Oriented Formal Specification and Refinement, http://www.eschertech.com/products/, last visited: February 2011.

[7] Samson W. B., Nevill, Dugard, Predictive software metrics based on a formal specification. In: Information and Software Technology. Volume 29, Issue 5, June 1987, pp. 242 – 248.

[8] Juei Chang and Debra J. Richardson. Static and Dynamic Specification Slicing. Technical report, Department of Information and Computer Science, University of California, 1994.

[9] Bollin Andreas, "Slice-based Formal Specification Measures - Mapping Coupling and Cohesion Measures to Formal Z." In: C. Munoz (Editor): Proceedings of the Second NASA Formal Methods Symposium. Hanover (MD): NASA Center for Aerospace Information (CASI), April 2010, pp. 24-33.

[10] Guide to the Software Engineering Body of Knowledge, IEEE, 2004

[11] Bollin Andreas, Specification Comprehension – Reducing the Complexity of Specifications. PhD thesis, Universität Klagenfurt, April 2004.

[12] Webster and Watson, 2002 J. Webster and R.T. Watson, Analyzing the past to prepare for the future: Writing a literature review, MIS Quarterly 26 (2002) (2), pp. 13–23.

[13] McCabe T.J., "A Complexity Measure," IEEE Transactions on Software Engineering, vol. 2, no. 4, pp. 308-320, July 1976.

[14] Rick Vinter, Martin Loomes, and Diana Kornbrot. Applying software metrics to formal specifications: A cognitive approach. In 5th International Symposium on Software Metrics, pages 216–223, Bethesda, Maryland, 1998. IEEE Computer Society.

[15] Juan C. Nogueira, Luqi, Valdis Berzins, and Nader Nada. A formal risk assessment model for software evolution. In Proceedings of the 2nd International Workshop on Economics-Driven Software Engineering Research (EDSER-2), 2000.

[16] Linda M. Laird & M. Carol Brennan. Software measurement and estimation, Wiley-interscience and IEEE computer society publishing, 2006.

[17] Peter Kokol, Vili Podgorelec, Henri Habrias, and Nassim Hadj Rabia. The complexity of formal specifications - assessments by alpha - metric. ACM SIGPLAN Notices, 6:84–88, 1999.

[18] Kuo-Chung Tai. A program complexity metric based on data flow information in control graphs. Proceedings of the 7th International Conference on Software Engineering, pages 239–248, 1984.

[19] David Carrington, David Duke, Ian Hayes, and Jim Welsh. Deriving modular designs from formal specifications. In ACM SIGSOFT Software Engineering Notes, volume 18, pages 89–98. ACM, December 1993.

[20] Arun Lakhotia. Rule-based approach to computing module cohesion. In Proceedings of the 15th International Conference on Software Engineering, pages 35–44. IEEE Computer Society Press, 1997.

[21] I. Sommerville, Software Engineering, 5th Edition, Addison Wesley, 1996.

[22] B. Boehm, C. Abts, and S. Chulani, "Software Development Cost Estimation Approaches—A Survey," Annals of Software Eng., vol. 10, pp. 177-205, 2000.

[23] Kemerer, C.F. An empirical validation of software cost estimation models. CACM, 30, 5 (May 1987), 416- 429.

[24] Putnam, L. and W. Myers, Measures for Excellence, Yourdon Press Computing Series, 1992.

[25] Boehm, Barry W., "Software Engineering Economics," Software Engineering, IEEE Transactions on , vol.SE-10, no.1, pp.4-21, Jan. 1984

[26] Musilek, P.; Pedrycz, W.; Nan Sun; Succi, G., "On the sensitivity of COCOMO II software cost estimation model," Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on , vol., no., pp. 13- 20, 2002

[27] Lionel C. Briand, Khaled El Emam, Dagmar Surmann, Isabella Wieczorek, Katrina D. Maxwell, "An assessment and comparison of common software cost estimation modeling techniques," Software Engineering, International Conference on, p. 313, 21st International Conference on Software Engineering (ICSE'99), 1999

[28] Jorgensen, M.; Shepperd, M., "A Systematic Review of Software Development Cost Estimation Studies," Software Engineering, IEEE Transactions on Software Engineering, vol.33, no.1, pp.33-53, Jan. 2007

[29] Bollin Andreas: Concept location in formal specifications. In: Journal of Software Maintenance and Evolution - Research and Practice, Hoboken (NJ): John Wiley & Sons Inc (2008), pp. 77-105

[30] Interpreting Regression Output, Princeton University website, http://dss.princeton.edu/online_help/analysis/interpreting_regression.htm, Last visited: June 2011

[31] S. Dowdy et al, Statistics for research, 3rd Edition, Wiley, 2004

The endless cycle of idea and action,

Endless invention, endless experiment,

Brings knowledge of motion, but not of stillness;

Knowledge of speech, but not of silence;

Knowledge of words, and ignorance of the Word.

Where is the Life we have lost in living?

Where is the wisdom we have lost in knowledge?

Where is the knowledge we have lost in information?

T.S. Eliot