UNIVERSITY OF CALGARY

The Effect of Amount of Software Reuse on Defect Severity in Real-Time C-Base Environment

By

Sheng Ouyang

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE

DEGREE OF MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

CALGARY, ALBERTA

MARCH, 2000

Abstract

An empirical investigation on the effects of software reuse on the severity of software defects is presented in this thesis. The results for the empirical analysis came from three real-time projects from a North-American telecommunications company.

Unlike the results from many published papers, the analysis results from this thesis do not show convincing evidence of software reuse benefits. Strong correlation is lacking between the amount of software reuse and all levels of software defects severity. The only exception is that the number of level 3 severity of software defects shows a strong negative correlation with reuse level.

The empirical study of this thesis suggests that software reuse will not decrease defect density at the different severity classes in every project. As a result, a practical question to be addressed in the future studies is in what kind of project will there be a guarantee of measurable benefit as a result from software reuse.

Acknowledgements

A thesis takes quite some time and effort to write.

I must thank my beautiful girlfriend and my family for their understanding and support.

I would also like to thank my two supervisors, Dr. Mike Smith and Dr. Giancarlo Succi, for their patience and helpful comments.

Table of content

List of Tables

List of Figures

List of Abbreviations and Acronyms used in this thesis

DD	Defect density
DD_i	Defect Density at severity class *i*
ERL	External Reuse Density
ERF	External Reuse Frequency
ERL	External Reuse Level
HV	Halstead Volume
LOC	Lines Of Code
MCC	McCabe Cyclomatic Complexity
RCR	Relative Cost of Reuse
RCWR	Relative Cost of Writing for Reuse
RD	Reuse Density
RF	Reuse Frequency
RL	Ruse Level

1 Introduction

1.1 Objectives

Software reuse is the process of implementing or updating software systems using existing software assets (Department of Defense, 1996). Software reuse can be applied to any phase of the project, such as design, coding, and testing stages. The types of reuse can be with data, documents, source code and so on.

Poulin (1997) has said that software reuse can increase software reliability and quality and that, in the long run, it can decrease the development and maintenance costs of a software system. However, there is currently very little empirical validation of this commonly held belief.

There are studies relating software reuse to the number of defect reduction (Frakes and Succi, 1997), the defect density reduction (Succi and Benedicenti, 1998), and the productivity improvement (Melo *et al.*, 1996). There are also studies relating reuse to time to fix defects and to phases in which defects have been introduced and detected (Thomas *et al.*, 1995). However, the effect of the amount of software reuse on defect density for the various severity classes has never been analyzed. This thesis is a contribution to overcome this lack of knowledge. Three real-time telecommunication applications projects of a North American multinational company were analyzed. Defects are classified in different classes according to their severity. Effects of amount-of-reuse are measured by reuse density, reuse frequency, and reuse level. The effects of amount-of-reuse on the different defect classes are studied in the following chapters. In this thesis, we attempt to accept or to reject the following hypothesis.

1. There is an inverse correlation between amount-of-reuse and defect density.
2. There is an inverse correlation between amount-of-reuse and defect density at different severity classes.

The independent and dependent variables in the metrics for analyzing Amount-of-Reuse and Software quality will be discussed in detail in later chapters.

1.2 Motivations

Software reuse process is a relatively new field. There are a lot of 'myths' about software reuse (Tracz, 1994), such as code reuse results in huge increases in productivity or that reused software is the same as reusable software. Therefore there is a need for more data to verify the effects of software reuse on different areas of software process. Many of the reuse data and reuse metrics are related to cost and benefit of software development.

1.2.1 Cost and benefits of software reuse

The goal of reusing software in a business is to save money. The money can be saved from shortened development time, through higher productivity, to lower maintenance costs. Since saving money is an important factor in the decision for developing a software reuse practice, it is important to know the potential cost of producing reusable components and the potential saving from reusing those components.

In general, reusing a software component takes less effort than developing the same component from ground up. However, there is still a cost associated with software reuse. A reusable component usually takes more effort to develop. The extra cost of writing for reuse may result from the production of better documentation or from doing better testing. Beside the extra cost of developing reusable software, there are still other costs relating to reusing a software component, such as the extra time required to search for the reuse function.

1.2.1.1 Relative-cost-of-reuse (RCR)

The relative-cost-of-reuse (***RCR***) is the portion of the effort that it takes to reuse a component without modification versus writing it from scratch (Poulin, 1997). It is generally a lot cheaper to reuse components already developed by other people. However, there are still costs associated with reusing software. The costs of reuse mainly come from understanding and integrating various reusable components into existing projects. The costs vary from one project to another, depending on the complexity of the project. Usually, the more complex a system is, the more difficult it is to integrate reusable components. The relative-cost-of-reuse can be estimated from one's own past experience or from a similar published project. In general, the relative-cost-of-reuse is about 0.20 (Poulin, 1997).

1.2.1.2 Relative-cost-of-writing-for-reuse (RCWR)

Software reuse is not a silver bullet. In general, it is more expensive to develop a reusable component than a non-reusable component. The additional costs of making a software reusable are being broken down as below (Tracz, 1994).

25%	**For additional generalization**
15%	**For additional documentation**
10%	**For additional testing**
5%	**For library support and maintenance**
60%	**Additional cost of making something reusable**

Figure 1.1: Relative-cost-of-writing-for-reuse

The relative-cost-of-writing-for-reuse (***RCWR***) varies from one type of project to another. Like everything else, the more complex a component is, the more expensive it is to design for reuse. In general, writing reusable software takes 60% additional cost to developed as shown in Figure 1.1. The data related to RCWR can come from one's past experience, or from the public domain.

1.2.2 The need for more software reuse data

Most of the software reuse papers published are on the successful results of various software reuse projects:

- A study done on 8 medium-size management information systems by the University of Maryland shown that productivity increases by 20 LOC per hour for every 10% reuse rate increase (Melo *et al.*, 1995).
- An analysis done on 8 medium scales Ada projects in the NASA's Goddard Space Flight Center shows 88% reduction in rework cost using verbatim reuse components (Thomas *et al.*, 1995).
- Frake and Succi (1997) found a strong positive correlation of 0.76 between software quality on a 10 point scale (10 = best) and amount of reuse on four industrial sites.
- Gaffney and Cruickshank (1992) found a strong positive correlation of 0.99 between percentage of code reuse and productivity (LOC/labor-month) in 50 different development sites.
- An analysis done on the student projects in the University of Maryland showed a strong negative correlation of –0.62 between reuse ratio and error density (Devanbu *et al.*, 1996).

There are few published software reuse failure cases. One of the high profile software reuse failure cases is the blow up of the European's Ariane 5 rocket in 1996 (Lion, 1996). Part of the software used in the inertial reference system in Ariane 5 rocket was reused from the earlier version of Ariane 4. The reused code crashed after the horizontal velocity of the new rocket exceeded its 16-bit integer storage. The blow up of the Ariane 5 rocket caused the European space agency billions of dollars in the loss of hardware and business opportunities.

One can not guarantee that software reuse always saves money because of the extra costs (*relative-cost-of-writing-for-reuse*) and benefits (*relative-cost-of-reuse*) of software reuse. However, there are few reports related to the failures of the software reuse

program, i.e., the failure of the Araine 5 rocket (Lion, 1996). One has to wonder that if software reuse is really the magic bullet as suggested in most published papers, or is more data is needed. An empirical study on three real-time real industrial projects will be done in the following chapters to see if software reuse really decreases software defect density. Lower defect density may correspond to lower development costs.

1.3 Reasons for choosing the current projects

Software systems are continually getting more complex and more expensive to build. Therefore more and more companies are trying to use various software reuse techniques to increase productivity, to reduce development cost, and to improve their software quality (Poulin and Caruso, 1992). However, software reuse is not for everyone. The success of reuse is dependent on many factors, such as low overhead cost, favorable software domain and management support for reuse (Isoda, 1996). Therefore, it is important to choose the right project to analyze software reuse. The three projects studied in this thesis have several factors that should contribute to the success of a software reuse project (Frakes, 1991).

- ***Narrow and well-defined domain***: The reuse components should be targeting a narrow and well-defined domain. In this thesis, the domain was a C-base real time communication environment. Usually, the narrower a domain is, the more likely a component will be reused. Reusable components in a narrow domain are not designed to do everything. They are only intended for use in a particular environment. Therefore, the components are usually less complex, cheaper to write, and easier to reuse. As a result, the overhead cost of producing and using reusable components will be lower in a narrow, well-defined domain project.
- ***Slowly changing domain technology***: A fast changing domain technology means a rapidly decaying usefulness of the reusable components. In some special cases, the development cost for reuse is higher than the saving due to the fast depreciation time. For example, it hardly makes any business sense to create reusable components for a dying product. Usually, the slower the domain

technology changes, the longer the recapture time is and the cheaper the reuse components are. The current technology domain of the case study did not change rapidly. First, the three analyzed projects were developed in 'C' language, not some old dying language. Secondly, the development cycle is very long, four years from beginning to finish.

- ***Economies of scale in market***: Building a reusable component is expensive. Therefore, reusable components are built when they are going to be reused frequently. In software cases, a reusable component is often a function that is being used over and over again as a building block of another function. Therefore, it makes sense to spend more effort in developing components that are more likely to be reused. The current product in the case study has more than one variant, and is going to have even more variants in the near future. Therefore, the economic scale is good. It makes sense to analyze and to develop reusable components for later reuse.
- ***Type of reuse***: There are two types of reuse, reusing a component with modification and reusing a component without modification. Reusing a component without modification is a lot better than reusing a component with modification. This is because the developer does not have to spend time to modify old components and to test the new modified components. Due to the difficulty of collecting information on modified components, only verbatim reuse of components is going to be analyzed in this thesis.

1.4 Organization of the thesis

The thesis is organized as follows.

- **Chapter 2**: Current software reuse and software quality metrics are introduced in this chapter. Special attention is given to discuss one of the software reuse metrics, amount-of-reuse, and its relationship between software quality and software development costs.
- **Chapter 3**: Five software attribute measurements used in the correlation analysis and linear regression are discussed in this chapter. The five software attributes are

lines-of-code, Halstead's Volume, defect severity, McCabe cyclomatic complexity of a program and amount-of-reuse measures.

- **Chapter 4**: Various software analysis techniques used in the thesis are introduced. The analysis techniques include simple descriptive techniques, graphical representations, and correlation and regression analysis.
- **Chapter 5**: An empirical investigation on the three projects is done in this chapter. The empirical investigation consists of context study of the projects, description and analysis of the statistical results, development of a linear model and discussion of the results.
- **Chapter 6**: Bootstrapping methods and the bootstrapping analysis of the results of this thesis are presented in this chapter. The bootstrapping analysis is intended to improve the analysis results from Chapter 5.
- **Chapter 7**: Summaries of the thesis and the ongoing research are discussed in this final chapter.

2 The current Software Reuse and Software quality metrics

2.1 Introduction

There are many stated benefits to reuse software (Poulin and Caruso, 1992), such as the increase in software reliability and quality. However, there are few published results on software reuse. Three C-based projects will be analyzed in this thesis in order to provide more data on the effects of software reuse.

The current metrics on software reuse and software quality are presented in the current chapter. First introduced are various metrics to measure software reuse. One of the software reuse metrics, amount-of-reuse, is then discussed in more detail. Next discussed are various software models used to analyze the relationship between amount-of-reuse and software quality, and the relationship between amount-of-reuse and software development costs. Finally discussed are various software quality metrics that can be used to analyze the effects of the amount of software reuse.

2.2 Current software reuse measurement

There are currently many metrics in use to measure and to analyze software reuse. Frakes and Terry (1996) use five different types of metric to analyze software reuse: cost-benefit analysis, maturity assessment, amounts of reuse, failure model analysis, reusability assessment, and reuse library metrics. In general, most of the commercial software reuse metrics are in one way or another related to the cost of software reuse. Some of the metrics for measuring software reuse are listed in Table 2.1.

Metric	Definition
Project-level's Return on investment - **ROI** (Poulin and Caruso, 1992)	The sum of cost saved through reuse minus the additional development cost.
2.2.1.1.1 Corporate-level's Return on investment – **ROI** (Poulin and Caruso, 1992)	2.2.1.1.2 The sum of the net present value on each of the project level ROI over the years.
Defense information system agency's Return on investment - **ROI** (Poulin and Caruso, 1992)	Effort of developing the project with all verbatim (without modification) code minus the actual cost of development using different types of reuse code (verbatim, modified).
Net Cost Saved – **NCS** (Poulin and Caruso, 1992)	The sum of the development cost of the reusable components minus the sum of the expected cost that the producer incurs in producing the reusable components.
2.2.1.1.3 Quality of Investment – **Q** (Poulin and Caruso, 1992)	The ratio of reuse benefit to the consumer vs. the reuse investment by the producer.
Benefit investment – **BI** (Poulin and Caruso, 1992)	The ratio of cost of project without reuse minus actual cost of project with reuse vs. cost of producing reusable components.
Break-even point - **N** (Gaffney and Cruickshank, 1996)	The relative cost of creating reusable code vs. the relative saving resulted from integrating the reused code.
Amount of Reuse - **Reuse%** (Poulin *et al.*, 1993)	The percentage of Reused verbatim Source Instructions from external repositories (RSI) to the sum of RSI and shipped source instructions (SSI).

Table 2.1: Current software reuse metrics

Each of the above software reuse metrics are calculated using different types of data. One can not say that one type of reuse metric is superior to another type of reuse metric because each of the metrics measures software reuse from different angles. There are two requirements in choosing a suitable software reuse measurement in this thesis:

- The data can be obtained through this company's database or from its software programs.
- The data are objective, and can be collected through an automated process.

Based on the above criteria, the metrics in Table 2.1 are reviewed. A key issue in deciding the type of metric used in this thesis is whether the required data are available

from the database of the telecommunication company whose process is analyzed in this thesis.

- Project-level's ROI (Poulin and Caruso, 1992) requires the calculation of various software related costs. The classification of costs is not objective since Poulin and Caruso do not provides a detail definition of the costs. Furthermore, the company database does not contain any information related to the cost of the project.
- Corporate-level's ROI (Poulin and Caruso, 1992) also requires the calculation of cost which is currently not available from the company's database.
- Defense information system agency's ROI (Poulin and Caruso, 1992) also requires the calculation of cost, which is currently not available.
- Net Cost Saved (Poulin and Caruso, 1992) requires the estimation of the number of times that an item is going to be reused in the future. Such estimation is subjective.
- Quality of Investment (Poulin and Caruso, 1992) requires data related to the reuse investment by the producer and to the reuse benefits to the consumer. Neither is currently available in the company's database.
- Benefit investment (Poulin and Caruso, 1992) also requires the calculation of cost, which is currently not available.
- Break-even point (Gaffney and Cruickshank, 1996) requires the knowledge of various relative reuse costs, which are currently not available.
- Amount-of-Reuse (Poulin *et al.*, 1993) metric requires the calculation of the number of instructions in a reused component and the size of the project. Even though the data on the lines of code is not available in the company's database, the LOC can be retrieved automatically by a software program. Currently, there is a software program called WebMetrics tool (Succi *et al.*, 1998) that can retrieve the data needed in the amount-of-reuse calculations.

As was shown in Table 2.1, different types of reuse metrics require different types of data in their calculations. As shown by the above analysis, the company's database contains only enough information to calculate one of the metrics, amount-of-reuse metric,

discussed in Table 2.1. This thesis will use amount-of-reuse metric to calculate amount of reuse after the projects were finished.

2.3 Amount of reuse

Different classes of measures exist for software reuse (Frakes and Terry, 1996). "Amount-of-reuse" is one of the software reuse measurements. The "amount-of-reuse" metric represents "how much" reuse is inside a given software system. In general, the amount-of-reuse metric in a system is the ratio between the amount software objects reused and the total size of software objects (Poulin, 1997). The objects can be files, functions, etc. The size of the objects can be measured in LOC, Halstead Volume (*HV*) (Halstead, 1977), etc. There are many ways to implement the amount-of-reuse, each of which provides a different aspect, ranging from how much code is reused to how often it is reused. Multiple measures of amount-of-reuse provide different perspectives on how people reuse components, and allow for a more balanced and complete picture of the effects of reuse (Ferri *et al.*, 1997).

Some of the proposed amount-of-reuse measures from various literatures are listed in Table 2.2.

Metric	Definition
Fraction New and Extensively Modified Components -**FNEMC** (Agresti and Evanco, 1992)	Percentage of new or extensively modified compilation units over total number of compilation units.
Reuse Percent -**Reuse%** (Poulin *et al.*, 1993)	Percentage of Reused verbatim Source Instructions from external repositories (RSI) to the sum of RSI and shipped source instructions (SSI).
Reuse Level –**RL** (Frakes and Terry, 1994)	Percentage of different items reused verbatim more than a given number of times vs total items used.
Reuse Frequency -**RF** (Frakes and Terry, 1994)	Percentage of references to items reused verbatim more than a given number of times vs total references.
Application Reuse Percentage - **AppReusePerc(S)** (Chen *et al.*, 1995)	Ratio of Reuse(S) to Size(S), where S is the system. Reuse(S) is the sum of the entities that S refers to.
Reuse Rate -**RR** (Basili *et al.*, 1996)	Percentage of code reused verbatim or slightly modified within the system.
Reuse Size and Frequency -**Rsf** (Devanbu *et al.*, 1996)	Percentage of code savings from reusing verbatim existing components vs. expected total size of the code with no external reuse and just one internal use per item defined.
Reuse Module size and Frequency –**RMsF** (Sadahiro Isoda, 1996)	The LOC of a reused module, and the times that it is being reused
Proportion of Reuse, **R** (Gaffney and Cruickshank, 1992)	Ratio between the lines of code reuse in an application and the total lines of code of the application.
Reuse Density -**RD** (Benedicenti *et al.*, 1997)	Normalized number of items reused verbatim more than a given number of times.

Table 2.2: General Amount-of-reuse metrics

The above metrics examine at the amount-of-reuse from different points of view. Each of metrics uses different types of data during calculation. There are two requirements in choosing an amount-of-reuse metric in this thesis:

- The result is objective; that is that the calculation can be done automatically by a tool called WebMetrics tool (Succi *et al.*, 1998).

- The calculation does not require data that are not available in the company's database.

Based on the above criteria, the amount of reuse metrics in Table 2.2 are reviewed:

- The **FNEMC** metric (Agresti and Evanco, 1992) requires the knowledge of "extensively modified" code, which is not available in the company's database and can not be measured automatically.
- The **Reuse%** metric (Poulin *et al.*, 1993) needs to have the data on the library source code and the repository code in order to separate the new code from the modified code and new code from the reused code. Unfortunately, the information on the library repository codes are not in the company's database and can not be calculated by the WebMetrics tool (Succi *et al.*, 1998).
- The **RL** and **RF** metrics (Frakes and Terry, 1994) use the verbatim items in the project for the calculation. The information on the C-functions can be collected by the WebMetrics tool (Succi *et al.*, 1998) automatically.
- The **AppReusePerc(S)** metric (Chen *et al.*, 1995) is similar to the **Reuse%** metric that needs to have data on the library source code and the repository code which are currently not available.
- The **RR** metric (Basili *et al.*, 1996) requires the knowledge of "slightly modified" code, which is not available in the company's database and can not be measured automatically.
- The **Rsf** metric (Devanbu *et al.*, 1996) requires knowledge of the size of code without reusing any functions. This requires the calculations of the size of all the library functions which can not be done by WebMetrics tools (Succi *et al.*, 1998).
- The **RMsF** metric (Sadahiro Isoda, 1996) is like the **Rsf** metric that needs to know the size of all the library functions which can not be done by WebMetrics tool (Succi *et al.*, 1998).
- The **R** metric (Gaffney and Cruickshank, 1992) needs to have the data on the library source codes which are not in the company's database and can not be calculated by the WebMetrics tool (Succi *et al.*, 1998).

- The **RD** metric (Benedicenti *et al.*, 1997) requires the knowledge of verbatim C-functions; which can be collected by the WebMetrics tool (Succi *et al.*, 1998) automatically from the database.

The company's database only provides data to calculate reuse level, reuse frequency and reuse density. Any other amount-of-reuse metrics need to use data that is currently not available. Furthermore, reuse level, reuse frequency and reuse density can be calculated by using a program called WebMetrics tool. As a result, it was decided that RL, RF, and RD were the best metrics to measure the amount-of-reuse metrics in this thesis.

2.4 Modeling the Effects of Amount-of-Reuse

An effective application of software reuse requires a quantified model of its effects. A general software reuse model has not yet been defined. However, several project specific models have been identified that link attributes of the amount-of-reuse to attributes of quality and productivity. Table 2.3 lists some of the project-specific models that have been identified in the past few years.

Reference	Independent variables	Dependent variables	Structure of model	Findings
(Agresti and Evanco, 1992)	FNEMC	Defect density	Log-Linear model	Higher levels of FNEMC result in higher defect density
(Basili *et al.*, 1996)	Reuse rate	Defect density, rework, and Productivity	Linear model with non-parametric testing	Higher reuse rates are correlated with higher productivity, lower defect density, and numbers of rework.
(Gaffney and Cruickshank, 1992)	Reuse%	Labor-month per function points, and unit cost	Linear and log-linear model	Larger reuse percent corresponds to lower development unit cost and smaller labor-month per function point
(Devanbu *et al.*, 1996)	Rsf	Error density, percentage of rework, and productivity	Log-linear model	Larger reuse size and higher reuse frequency correlate to lower error density and higher productivity.
(Frakes and Succi, 1997)	RL, RF	Software quality rating	Linear model with Non-parametric testing	Higher reuse level and reuse frequency corresponds to higher quality ratings and lower levels of defects and deltas.
(Succi and Benedicenti, 1998)	RD	Density of customer complaints	Linear model with Non-parametric testing	Higher reuse density corresponds to lower customer complaint density

Table 2.3: Effects of amount-of-reuse models

As shown in Table 2.3, that there are many studies relating the effect of the amount-of-reuse. However, the effect of software reuse on defect density levels for *various severity classes* has never been analyzed. In order to overcome this lack, it is proposed to analyze the relationship between the amount-of-reuse and defect-density levels for various severity classes. Defect-severity is one of the indicators for software quality. By introducing different software quality metrics in the following chapter, defect-severity is

shown to be the most suitable metric to measure software quality based on the available data in this thesis.

2.5 Introduction to software quality metrics

Software reuse can be used to improve the quality of software. There are many ways to define software quality. Different people define quality according to their own special needs. William defines software quality as the degree to which the software processes desired attributes, such as reliability, maintainability, flexibility and so on (William *et al.*, 1992). All those software quality aspects can be measured by different metrics. Reliability is chosen as the measurement for software quality in this thesis. Reliability is classified as software quality in this thesis. This chapter is intended as a background discussion on various current software quality metrics. By highlighting the strengths and limitations of each quality metric, defect density is shown as the most suitable metric to measure software quality for this thesis.

2.5.1 Software quality metrics

In the eyes of many managers, software quality directly translates to development cost, time to market, or customer response time. One example of the time related metric is called *spoilage* (Fenton and Pfleeger, 1996). Spoilage is very popular among Japanese companies, and is calculated as below:

$$system\_spoilage = \frac{time\ to\ fix\ post\ /\ release\ defects}{total\ system\ development\ time}$$

Currently, there is no information on the actual time spent to fix errors in the company's database.

Besides using time metrics, software quality can also be measured by its cost. The example of a cost related quality metric is the calculation of portability (Fenton and Pfleeger, 1996).

$$portability = 1 - \frac{resource\ needed\ to\ move\ system\ to\ target\ envirenment}{resource\ needed\ to\ create\ system}$$

Despite its benefits, the actual reuse cost is not easy to calculate. Furthermore, there is currently no information on cost in the company's database.

Most of the quality metrics measure software quality after a project is done. However, there are other types of metrics that can be used to predict the quality of a future project. One of such metrics is COQUAMO, sponsored by the European Strategic Program for Research in Information. COQUAMO estimates software quality based on the observation of quality in the previous projects implemented by the same organization (Kichenham, 1989).

Software quality can also be measured in terms of software complexity. Usually, a complex program has more errors than a simple program. Thus, a complex system often has lower quality. Software complexity can be measured by many metrics, such as McCabe cyclomatic complexity metric, Henry and Kafura's fan-in and fan-out measurement (Card and Agresti, 1998), or Hutchens and Basili's Data binding (Selby and Basili, 1991). Due the abilities of the WebMetrics tool (Succi *et al.*, 1998) to collect data on McCabe cyclomatic complexity, it is decided in this thesis to use McCabe cyclomatic complexity as one of the measurements for software quality. McCabe cyclomatic complexity will be discussed in more detail in Chapter 3.4: *Software complexity attributes*.

Due to the shortcomings (time estimation, resource estimation, or software quality prediction) of the various software quality metrics discussed above, defect-density was chosen as another software quality measurement in this thesis. There are many benefits in using defect density metric. First, defect-density is one of the most widely used quality metrics. Next, defect-density is a direct measurement of software quality. Moreover, defect-density can be used in every stage of the development process. The data needed to calculate defect-density are in the company's database, or are available through the WebMetrics Tool (Succi *et al.*, 1998).

Defect density of a systems *S*, *DD(S)*, is defined as the ratio between number of defects (of a given class) in *S* and the size of *S*:

$$DD(S) \equiv \frac{Number\ of\ Defects\ in\ S}{Size\ of\ S}$$

Defects can be classified according to different perspectives, such as the origin, the cost to fix, how severely they affect the behavior of the system, and so on. The company provided the data for this thesis chose to classify defects into different classes of severity.

Defect density can be specialized to specific classes of severity. The "Defect Density" for a given severity class *i* (DD_i) is calculated as follow:

$$DD_i(S) = \frac{Number\ of\ Defects\ of\ Severity\ i\ in\ S}{Size\ of\ S}$$

Clearly the summation over all the defect density of the different severity classes is the global defect density:

$$\sum_i DD_i(S) = DD(S)$$

The classifications of the defects in this thesis are discussed in Chapter 3.3: *Software Quality Attributes*. The defects are classified according to five different types of severity in this thesis. The product size used in the defect density at the different severity classes (DD_i) calculations is lines of code. The defect density metric will be used in the later chapters to analyze three C-based projects.

2.6 Summary

Current software reuse measurement was introduced in the first section of this chapter of this thesis. By comparing weaknesses from various software reuse metrics, the amount-of-reuse metric was chosen as the best one for measuring software reuse. Next, various types of amount-of-reuse metrics were discussed. Reuse density, reuse frequency and reuse level are chosen as the metrics for measuring amount-of-reuse because of the limitations of the company's database. Finally, based on the software quality metrics discussion, it was decided to use defect severity and McCabe cyclomatic complexity as software quality measurements in this thesis.

3 Software attributes measurement

3.1 Introduction

Software metrics can be used to collect and to analyze various kinds of software project data. The properties of data are called attributes. It is important to have well-defined software attributes measurements in order to avoid comparing apples with oranges during analysis.

In the following sections of the current chapter, there are definitions of five different software attributes, which are used in various software reuse and software quality metrics later in the thesis. The five software attributes are lines-of-code, Halstead's Volume, defect severity, McCabe cyclomatic complexity of a program and amount-of-reuse measures.

3.2 Size of components attributes

3.2.1 Lines Of Code

Software reuse and software quality measurements both contain the size of component attributes. There are many ways to measure the size of a program, ranging from lines-of-code (LOC), memory required, to code size in Kbytes. LOC is the most commonly used among those measurements. There are many ways to measure lines-of-code. Therefore, it is important to understand how LOC is being measured in this thesis and in the industry, in order to avoid comparing the data with the same attributes, but with different definitions.

Before counting or comparing lines-of-code, it is important to know what type of language is being used in a program. The size of the code is greatly influenced by the type of language. Usually, assembler code has the largest LOC count, while fourth-generation language and visual programming have the lowest lines-of-code counts. Therefore, it is important to compare the LOC of one language of one program with the

LOC of the same language of another program. Three software projects written in C are analyzed in this thesis. It would be inappropriate to compare the results reported in this thesis with other projects developed in different languages.

After understanding the language type, the next step is to measure the lines-of-code in a program. The basic lines-of-code was calculated by simply counting the number of lines of the code in a source program. Therefore, the whole data collection process for lines-of-code can easily be reproduced and automated. There are many benefits in measuring LOC. For example, the size of design may provide an indication of the size of code, and the size of code may be used to predict the development cost of a project.

Despite the advantages of using lines-of-code as a software measurement, there are some weaknesses associated with this. First, there are many ways to count LOC. Lines-of-code can be counted by the number of instructions in a program. LOC may also contain the number of blank lines, comment lines, and declarations. As a result, different ways of counting LOC can produce very different results. In this thesis, LOC is measured by counting the number of semicolons in the source code (Humphrey, 1995). Moreover, blank lines, comment lines, and declarations are ignored in this thesis, as they are not considered as software reuse.

Lines-of-code is used in most of the software metric measurements. Therefore, it is important to know that there are different factors, which may affect the final counts of lines-of-code. Some of the factors are listed below (Fenton and Pfleeger, 1996).

1. Some instructions are harder to write than others.
2. The final program may contain dead code and test code.
3. One instruction may contain several sub-instructions.
4. Documentation also takes effort to write. Documentation is important for the understanding and maintenance of the software.

This thesis only considers code reuse. The size of software is measured by lines-of-code. The LOC calculation is based on the assumptions that all the instructions take the same amount of effort to write, and the program does not contain any dead code or test code.

3.2.2 Halstead Volume

Halstead Volume (*HV*) (Halstead, 1977) was conceived to approximate the amount of memory space required by a program. Helstead theorized that each program could be defined as a collection of tokens. Each token can be classified as either an operator or an operand (Mills, 1988). To compute *HV*, it is necessary to count the number of different operators (n_1) and operands (n_2) and the total number of operators (N_1) and operands (N_2) that appear in a program. The Halstead Volume *HV* is calculated as follow.

$$HV = (N_1 + N_2)\log_2(n_1 + n_2)$$

Studies have shown that the values of LOC and *HV* appear to be linearly related and equally valid as relative measures of program size (Mills, 1988).

3.3 Software quality attributes

Software quality means different things to different people. For example, customers may see the lack of software quality as the number of crashes a program has over a period of time, while designers may see the lack of software quality in the same program in terms of documentation quality as they are modifying the program. This thesis measures software quality in terms of *defects* discovered after the product is released from the programmers and is in the hands of the testing team.

Several definitions of "software defect" exist. Following the definition of Humphrey (1990), a defect is defined in this thesis as the *improper program conditions* that are generally the result of an error. Defects used in this thesis are detected by testers during

the testing phase of a system, that is, after the coding of the system has been completed and before the system is delivered to customers

Defect density levels for various severity classes are analyzed in this thesis. Chapter 2.5.1 contains general definitions of Defect Density for different severity classes (DD_i).

The defects are classified according to the definitions defined by the company, which provides the data used in this thesis. This company uses a 5 point ordinal scale to classify defect by severity, from 1 –the most serious defect, to 5 –the least serious defect. A generic definition of each point follows:

- *Severity 1*: unrecoverable failure of the software
- *Severity 2*: system operations can be recovered by powering down and powering up the system
- *Severity 3*: part of the system feature does not work as specified
- *Severity 4*: usability problem
- *Severity 5*: cosmetic deficiency

Once the defect severity is classified, the next step is to count defects according to the defect severity levels discussed above. The plain count of number of defects does not provide a comprehensive view of how well the system has been developed. In general, a larger system is usually more complicated. Thus, a larger system has a larger number of defects. Therefore, the notion of defect density was introduced (Fenton and Pfleeger, 1996). Defect density is the ratio between the number of defects in a system and the size of a system. Defect density can refer either to the whole set of defects or to specific classes of severity; both types of defect density were analyzed in this thesis.

3.4 Software complexity attributes

A software system can also be measured by its complexity in addition to lines of code and software defects. As with software quality, there are also many views on software complexity. Fenton defines two types of complexities, the complexity of a problem and

the complexity of a solution (Fenton and Pfleeger, 1996). Each type of complexity has its own measurements. For example, function points for measuring the complexity of a problem, and McCabe cyclomatic complexity for measuring the complexity of a solution. Software solution complexity, simply called the software complexity in the following chapters, is analyzed in this thesis.

There are many ways to measure the complexity of a solution. The complexity can be measured in turns of control-flow structure, data-flow structure and data structure (Fenton and Pfleeger, 1996). The control-flow structures approach measures the execution of the sequence of the instruction in the program. The data-flow structure approach follows the creation and the modification of a single data item. On the other hand, the data structure measures the internal makeup of a data item (object). The projects measured in this thesis do not have complicated data structures, or data handling features. Therefore, only control-flow structure is used to measure the software complexity.

The McCabe cyclomatic complexity is used to measure the control-flow structure of the program in this thesis. McCabe cyclomatic complexity is one of the most popular ways to measure complexity. Its unit is called *cyclomatic complexity of a program*. McCabe cyclomatic complexity is based on the flow chart. A flow chart consists of *nodes* and *edges*. A node is a software command, or a decision in the program. An edge is the line connecting one command to another structure (Fenton and Pfleeger, 1996). The McCabe cyclomatic complexity *MCC* is calculated as follow.

$$MCC = e - n + 2$$

where

e = edges

n = nodes

Example 1 is a small program in a single file.

```
// a small program file, files.c

extern int StudentMark;

     extern void output(char *string);
     extern void kickOutStudent();
     extern void award();
     int getMark() { return StudentMark);

     Main()
     {
          int FinalGrade = getMark();

          if (FinalGrade < 50)
          {
               output ("Fail");
               kickOutStudent ();
          }
          else
          {
               if (FinalGrade > 80)
                    output ("Honour");
               else
                    output ("Pass");
          }
          output ("Done");
          exit();
}
```

The flow diagram of example 1 is shown in Figure 3.1.

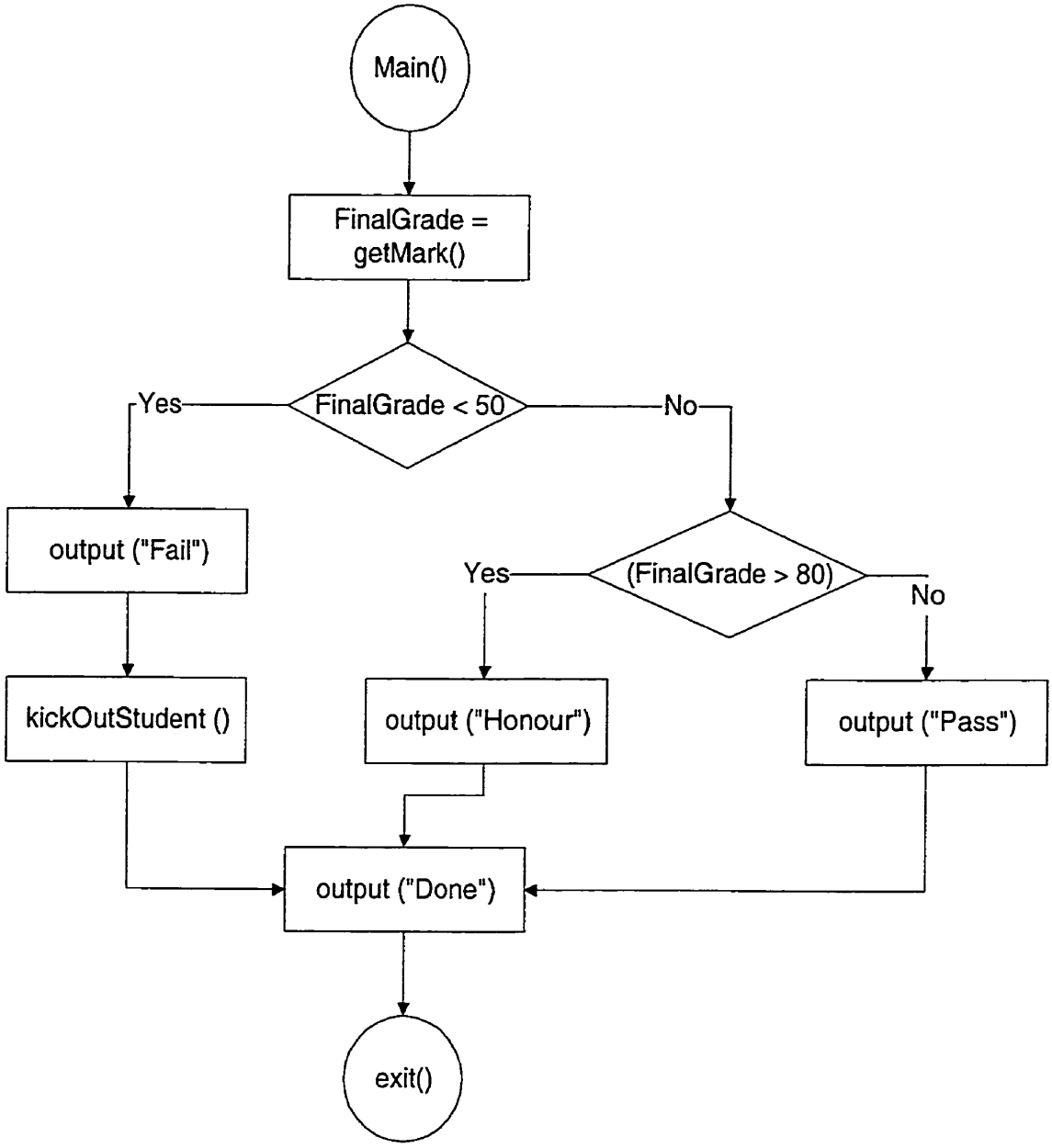

Figure 3.1: Flow chart for calculating McCabe cyclomatic complexity

Based on Figure 3.1, the $e = 11$, $n = 10$ and the cyclomatic number of example 1 is 3.

3.5 Amount-of-reuse attributes

Based on the discussions in Chapter 2.3, Reuse Frequency (Frakes and Terry, 1996), Reuse Level (Frakes and Terry, 1994) and Reuse Density (Benedicenti *et al.*, 1997) are used in this thesis to analyze the relationship between amount-of-reuse and software quality.

3.5.1 Reuse Frequency

Reuse Frequency (RF_t) is the ratio between the number of references to lower level items reused verbatim more than a given number of times (the threshold t) inside a higher level item (UF_t) and the total number of references to lower level items in the higher level item (TF) (Frakes and Terry, 1997):

$$RF_t = UF_t / TF$$

Only C code is used to analyze the amount of reuse in this thesis, where the reused lower level items are the C functions, the higher level items are the files. File is the unit of modularization in C in this thesis.

Reuse Frequency can be further divided into Internal Reuse Frequency (IRF_t) and External Reuse Frequency (ERF_t). This is done by separating in the computation of UF_t the reuse items coming from inside and from outside the working environment. Poulin (1997) considered internal reuse as a good programming practice. Following Poulin (1997) only reuse coming from outside the scope of the project was considered in this thesis: ERF_t in this case. Along the lines of (Frakes and Terry, 1994), the value of the threshold t is set to 2, that is, an item is considered "reused" if it used more than twice.

3.5.2 Reuse Level

Reuse Level (RL_t) is the ratio between the number of lower level items reused verbatim more than a given number of times (the threshold t) inside a higher level item (UL_t) and the total number of lower items in the higher level item (TL) (Frakes and Terry, 1997):

$$RL_t = UL_t / TL$$

The reused lower level items are the C functions in this thesis, while the higher level items are the files, the unit of modularization in C.

Reuse Level can be further divided into Internal Reuse Level (IRL_t) and External Reuse Frequency (ERL_t). This is done by separating in the computation of UL_t the reuse items coming from inside and from outside the working environment. Following Poulin (1997), only reuse coming from outside the scope of the project is considered in this thesis: Therefore, it is decided to only focus ERF_t. Along the lines of (Frakes and Terry, 1994), the value of the threshold t is set to 2.

3.5.3 Reuse Density

Reuse Density (RD_t) is the ratio between the number of references to lower level items reused verbatim more than a given number of times (the threshold t) inside a higher level item (UF_t) and the size of the higher level item (SI) (Benedicenti *et al.*, 1997):

$$RD_t = UF_t / SI$$

The same considerations made for Reuse Frequency and Reuse Level apply to Reuse Density. It was decided to use ERD_2, that is, External Reuse Density referring to items coming from outside the scope of the project and using 2 as the threshold. The size of the higher level item, the file, is measured in Lines Of Code (LOC), counting the number of semicolons in the source code (Humphrey, 1995).

3.5.4 Examples on Reuse Level, Reuse Frequency and Reuse Density

Table 3.1 provides a summary on the functions in example 1 in terms of internal or external function, and the number of references made to each function.

Function Name	Internal/External	References
Output()	External	4
KickOutStudent()	External	1
Award()	External	0
Exit()	Internal	1
GetMark()	Internal	1

Table 3.1: Classification of functions of example 1

Threshold	$ERL_{Threshold}$	$ERF_{Threshold}$	$ERD_{Threshold}$
0	2/4	5/7	5/12
1	1/4	4/7	4/12
2	1/4	4/7	4/12

Table 3.2: Amount-of-reuse of example 1

Table 3.2 provides a summary of the results of the amount-of-reuse of example 1. The detailed calculations of the results are as follows.

There are a total of 3 different types of functions (lower level items) used within higher level items (*file1.c*) of example 1. Inside the main function, there are 2 external and 1 internal functions used. The threshold 0 of ERL_0 is 2/4. Of the 2 external functions, only function Output(), is used more than once. Therefore, the results of ERL_1 and ERL_2 are 1/4.

There are a total of 6 references to the functions (lower level items) within *file1.c* (higher level items) of example 1. Function Output() is referenced four times, while function KickOutStudent() and function GetMark() are only referenced once. Therefore, the result of ERL_0 is 5/7. Of the two external functions, only function Output() is referenced more than once. Thus, the results of ERL_1 and ERL_2 are 4/7.

The size of the higher level item (*file1.c*) of example 1 is 11 lines of code by counting the number of semicolons in the source code (Humphrey, 1995). The same logic in the number of reference calculation for Reuse Frequency applies to Reuse Density. Thus, the result of ERL_0 is 5/12 and the results of ERL_1 and ERL_2 are 4/12.

3.6 Summary

The definitions of the five software variables used in later parts of this thesis were defined in this chapter. The five types of software variables are lines-of-code and Halstead Volume for measuring size of component, defect severity for measuring software quality, McCabe cyclomatic number for measuring software complexity and reuse level, reuse frequency and reuse density for amount-of-reuse measurement.

4 Statistics for software analysis

4.1 Introduction

With the raw data on a software process collected and categorized, statistical analysis can then be used to represent characteristics of the current data, and to predict trends in the data. Spreadsheet programs and statistical packages can be used to automatically analyze and graphically display the data. However, it is still important for users to understand the techniques available and to know when to chose an appropriate analysis techniques.

This chapter is intended as a tutorial to discuss the techniques used to analyze the data set in this thesis. First introduced are the various simple descriptive techniques used to represent statistical data associated with the population set from which the data is drawn. Discussed next are the statistical test techniques used to verify if a sample set indeed represents the population set together with graphical representations of the data characteristics. Finally there is a brief introduction to correlation and regression analysis applied in a software-engineering context.

4.2 Simple variable descriptive statistics

The basic descriptive statistical analysis techniques to numerically describe data sets are: mean, median, mode, quartiles, variance and standard deviation. These numerical representations provide a simple way of representing the data set. Several examples of using basic descriptive techniques to represent data are demonstrated with the simple data set of example 2.

> Example 2: Project C has 34 files. External reuse densities of threshold 2 (ERD_2) of each of the 34 files of project C are shown in Table A.1 in the appendix

The **mean** is the average of a data set. The mean is defined as the sum of all the data divided by the number of data point. The mean works the best in an evenly distributed data set. Any extreme values will influence how well the mean represents characteristics of the data set. The mean of example 2 is equal to 0.241. As the mean is close to the mid-point of the data set, and there are no real extreme values, the mean characterizes represents this data set well.

Another way to represent the data set is to use the **median**. Median is the middle value in an ordered set. In the above example, the median is 0.244. For the data set that has an even number of values, the median is calculated by averaging the two middle values. On the other hand, **mode** is the most commonly occurring values. Unlike mean and median, a data set can have more than one mode. Example 2 has three modes: 0.234, 0.272 and 0.283.

Mean and median are very useful in analyzing the characteristic of the set. However, they alone still can not represent the full picture of the data set under some circumstances as shown by example 3.

> Example 3 has two data sets, data set1 and data set2. Data set1 = (-4, -2, -1, 0, 1, 2, 4) and data set2 = (-400, -200, -100, 0, 100, 200, 400).

The values for the mean and median of example 3 are both equal to zero. However, these two data sets are quite different from each other. Therefore, one will need another way to represent the data that just using the mean and median.

The **quartile** is another statistical term used to represent the characteristics of an *N-point* data set by separating the data set into four equal parts. The first, or 25^{th}, quartile represents the data at the position of $(N + 1) / 4$. The second (50^{th}) quartile, almost identical to the median, represents the data at the position of $(N + 1) / 2$. The third (75^{th}) quartile represents the data at the position of $3 (N + 1) / 4$. The quartiles of example 3,

(-2, 0, 2) and (-200, 0, 200) describe the data sets better than the mean or the median. The quartiles not only show the difference, but also the dispersion of the data set. For example, the quartiles of example 3 show that 75% of the data are less than 2, and 25 % of the data are less than –2 in data set 1.

The **range** represents the data set from yet another angle. The range is the difference between the largest and the smallest value in the data set. Using the range easily highlights the difference between the data sets in example 3. The ranges in example 3 are equal to 8 and 800 respectively. Both quartile and range are used in the box plot diagram (discussed later) to help graphically visualize the distribution of the data set.

A limitation of the quartiles and range measure is that they are used to characterize the variations of the whole data set based on the characteristics from a few data points. Another type of variation measure called **variance** overcomes this weakness. Variance looks at the variation of the whole data set. Variance is the sum of the squared difference between each individual value (X_i) and the mean of the data (X_{mean}) over the total number of the data (N).

$$\sigma = \frac{\left((X_1 - \mathrm{X}_{\mathrm{mean}})^2 + (X_2 - \mathrm{X}_{\mathrm{mean}})^2 + \cdots + (X_N - \mathrm{X}_{\mathrm{mean}})^2\right)}{N}$$

$$= \frac{\sum (X_n - \mathrm{X}_{\mathrm{mean}})^2}{N}$$

The **standard deviation** is the square root of the variance. The advantage of standard deviation over variance is that the deviation has the same units as the original data. For example, the standard deviation shows the variation of data set1 is 100 times larger than the variation of data set2 in example 3. The standard deviations are 2.646 and 264.6 respectively.

The **kurtosis** measures the "peakness" of the bell-shape curve of the data distribution. Thus, the smaller the kurtosis, the flatter the bell curve is. In other words, a small kurtosis means the data is uniformly distributed, and a large kurtosis means that the data is concentrated around its mode. There are several methods for calculating the kurtosis. The kurtosis formula presented here is called the *coefficient of peakedness,* α_4 (Merrill and Fox, 1970).

$$\alpha_4 = \frac{1/n \sum_{i=1}^{n} (X_n - X_{mean})}{\sigma^4_{standard\ deviation}}$$

Skewness is used to measure the symmetry or lack of symmetry of a distribution. In a symmetric distribution, 50% of the data is on the left side of the mean and 50% of data is on the right side of the mean. As a result, the mode, the median and the mean are the same. In general, a large positive skewness means there are many large extreme data in the data set. On the other hand, a negative skewness means there are more small data than large data in the population. Zero skewness means a perfect symmetric distribution. There are several methods for calculating the skewness. The skewness formula presented here is called the *Pearsonian coefficient of skewness* (Parsons, 1974).

$$S_k = \frac{3\left(X_{mean} - X_{median}\right)}{\sigma_{standard\ deviation}}$$

4.3 Statistical tests

The basic descriptive methods described in the previous section do not show how well the data set actually represents the whole population. Usually, the data set is only a sample of the whole population. Therefore, there is a difference between the calculated characteristics of the known data and the characteristics of the actual data. As a result,

there is a need to do a statistical test to see how well the calculated result represents the real population data set.

Most of the statistical tests are based on the *central limit theorem* (Cangelosi *et al.*, 1983). The central limit theorem states that as the sample size grows larger, the sample's distribution gets closer and closer to the true distribution. In other words, the mean of a large sample size is close to the mean of the population regardless of the shape of the population. Based on the properties of the central limit theorem, statistical tests can infer the hypothesis, H_0, of the population from a sample of data.

There are five basic steps for doing the statistical test (Cangelosi *et al.*, 1983).

1. Formulate a null hypothesis about H_0 (equality hypothesis)
2. Formulate an alternative hypothesis (inequality hypothesis)
3. Select a level of significance
4. Perform the test to determine if a relationship really exists
5. Check the significance against the required significance. Accept the hypothesis of step 1 if the result is greater than the required significance. Otherwise, accept the hypothesis of step 2.

There are many tests that can be performed in step 4, such as parametric correlation test for normally distributed data sets (Frakes and Succi, 1997; Succi and Benedicenti, 1998), and the non-parametric correlation test for non-normal data set (Basili *et al.*, 1996). The test for normality is discussed in chapter 4.5.4.4. The amount and the characteristics of the data from the target company suggests the use of parametric correlations and linear models. Parametric techniques are robust to deviations from normality (Cochran, 1947); Briand *et al.* (1996) contains an in depth discussion on the use of parametric methods in software engineering.

4.3.1 Pearson's correlation

The Pearson's correlation can be applied to a normally distributed data set. In a normally distributed data set, residuals are normally distributed. A residual is the difference between the actual dependent value and the estimated dependent value obtained from using the linear regression formula. Pearson's correlation is also known as the *sample coefficient of correlation.* The sample coefficient of correlation r is calculated as follow.

$$r = \frac{N\sum X_n Y_n - \sum X_n \sum Y_n}{\sqrt{\left[N\sum X_n{}^2 - \left(\sum X_n\right)^2\right]\left[N\sum Y_n{}^2 - \left(\sum Y_n\right)^2\right]}}$$

where X_i and Yi are the values of the independent and independent values of the N point date set.

The interpretations of the coefficient of correlation are shown below.

$r = 0$: no correlation between the independent and "assumed" dependent variables what so ever

$r = 1$: complete positive correlation

$r = -1$: complete negative correlation

Pearson's correlation can provide an actual representation of the linear association between two variables in a large normal population set. However, in a small data set environment, the accuracy of the value of the coefficient of correlation is highly dependent on the normality of the data set. By removing a few of the extreme values in a small size data set can dramatically change the value or even the positive/negative sign of the coefficient of the correlation. There is a technique called bootstrapping that can be used to determine the range of validity of the correlation. The bootstrapping technique will be discussed in Chapter 6.

4.4 Graphical representation -- "A picture is worth a thousand words".

Despite the strengths of the number representations and statistical tests, a graphical representation can more easily convey information to readers. Many kinds of graphical representation can be used in data analysis. The most commonly used ones are box plots, histograms, and scatter plots.

4.4.1 Box plot -- also called the box-whisker plot

The box plot graphically displays the previously discussed descriptive data, median, quartiles, maximum, and minimum as shown in Figure 4.1

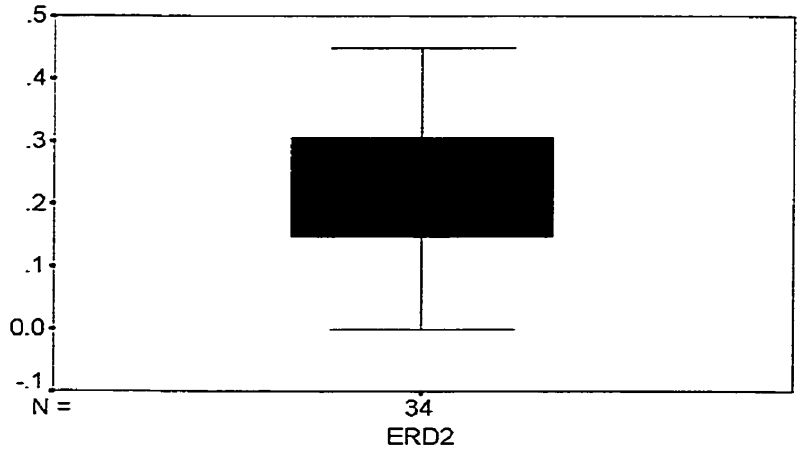

Figure 4.1: Box plot for ERD_2 of example 2.

The 'box' is bounded by the values of the 25th percentile and 75th percentile. The line drawn through the box represents the median. The area within the box is the theoretical place where all the data points are likely to be found if the distribution is normal. The 'whisker' lines are the lines from the 25th percentile to the minimum value and from the 75th percentile to the maximum value.

Currently, there are a number of different definitions for the minimum and maximum values. Some authors define them as the actual minimum and maximum values of the whole data set. On the other hand, in this thesis, the SPSS software definition is followed.

In this definition, the maximum value is taken as the 75^{th} percentile plus 1.5 times length of the box, and the minimum value as the 25^{th} percentile plus 1.5 times length of the box. Finally, the box plot uses asterisks to show the extreme values that are outside the whisker lines.

Figure 4.1 shows the box plot for example 2. The median, 0.244, is at the center of the box in Figure 4.1. The absence of any asterisks in the box plot shows there are no extreme values in the data set.

4.4.2 Normal Curve Histogram

Normal curve histogram is one type of the available bar plots. A bar plot consists of evenly spaced vertical bars on a horizontal line. The height of each vertical bar in the histogram corresponds to the number of data values that falls in a particular range in the data set. Frequency zero does not have a vertical bar. Histograms are used only when the variables in the data set can be meaningfully grouped. Unlike a basic histogram, a normal curve, calculated from the same data set as the histogram, is superimposed on top of the histogram used in this thesis. The formula for drawing a normal curve can be found in many statistic textbooks, such as the one by (Carlson and Thorne, 1997). The additional normal curve in a histogram helps to highlight the distribution of the data set. The normal curve can be calculated from the mean and the variance of the data set (Carlson and Thorne, 1997). Figure 4.2 below shows the histogram for example 2.

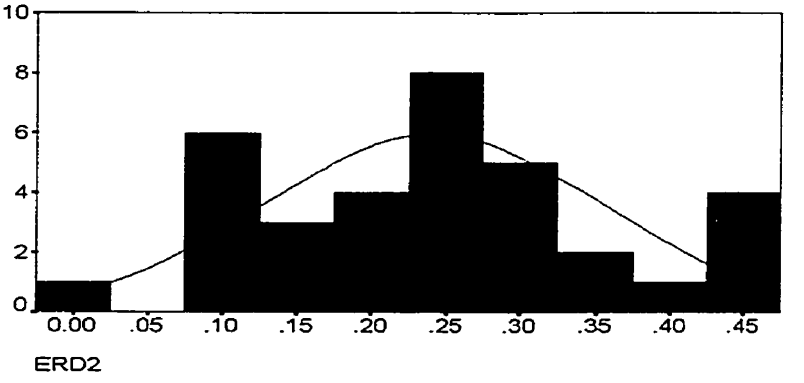

Figure 4.2: Histogram for ERD_2 of example 2.

Unlike the box plot where only the median and quartiles are used to represent the data set, more data values are used in Figure 4.2 to show the data distribution in finer details, and also displays the data set in a normalized bell curve. Despite its strengths, a normal curve histogram still can not display each individual value and the relationship between the variables.

4.4.3 Scatter diagram

Unlike the histogram, the scatter diagram is a graphical tool used to show the relationships of two types of variables, dependent and independent variables. The dependent variable is plotted on the Y-axis, usually called the '*y*' variable. The independent variable is plotted on the X-axis, called the '*x*' variable. The data points in the diagram are not connected together by a line.

A scatter plot shows the general trends of a system, i.e. increasing, decreasing or random. By looking at the points, one may be able to visualize if there is a relationship between dependent and independent variables, and if the relationship is positive or negative.

> Example 4: Project C has 34 files. Table A.2 in the appendix contains two types of project C data, threshold 2 external reuse density (ERD_2) and priority 3 of defect density (DD_3).

In Figure 4.3, the points are grouped in a manner that suggests a slight negative relationship between the threshold 2 external reuse density (ERD_2) and the priority 3 of defect density (DD_3) for example 4. The scatter diagram shows that the more external code reuse a file has, the less errors the file is going to have. Furthermore, the scatter plot shows an extreme data point of (0.059, 0.111). This extreme data point may affect the accuracy of the Pearson correlation value. The Pearson correlation value will be lower in this case.

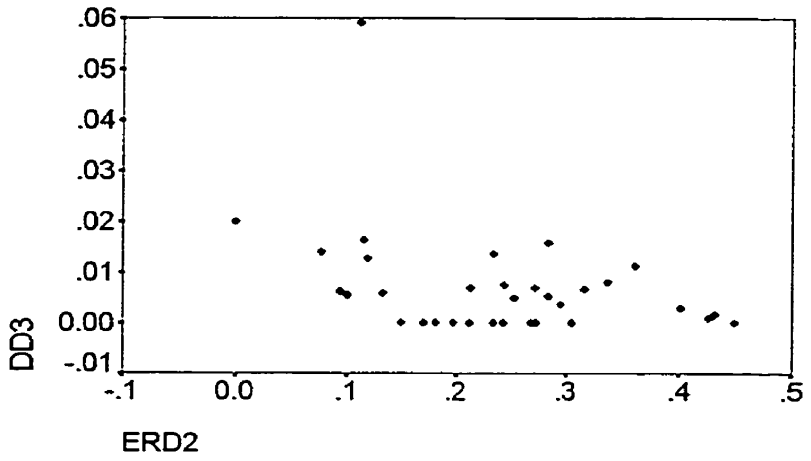

Figure 4.3: Scatter plot between ERD_2 and DD_3

4.5 Simple linear regression analysis

There are other types of analysis techniques that use a mathematical equation to represent a data set, and to predict the trend from the data set. Linear regression technique is chosen for use in this thesis to analyze the data set. Linear regression analysis provides an easy, yet effective way by attempting to represent a data set by using a mathematical equation, $Y = aX + b$.

There are also other benefits in using linear regression (Hamburg, 1970).

1. To provide estimates of values of dependent variables from the values of independent variables.
2. To obtain a measure of error involved in using the regression line as the basis for estimation. If the differences between the estimated values and the actual values are small, then the regression line can accurately represent the actual data set. However, if the differences are large, then the regression line can not accurately represent the actual data set.

The correctness of using linear regression on a data set is dependent on four hypotheses (Benedicenti *et al.*, 1997). The four hypotheses will be discussed in more details in Chapter 4.5.4.

1. **Linearity**
2. **Homoscedasticity**
3. **Normality**
4. **Independence of error**

There are four steps in doing linear regression:

1. Compute the correlation coefficient. (for more detail, see chapter 4.5.1)
2. Plot the scatter diagram. (for more detail, see chapter 4.5.2)
3. Compute the regression equation and its standard error of estimate of the dependent values, Y. (for more detail, see chapter 4.5.3)
4. Check the correctness of the linear regression function using the four hypotheses listed above. (for more detail, see chapter 4.5.4)

These 4 steps are illustrated in the following subchapters for verifying a negative relationship between the threshold 2 external reuse density (ERD_2) and the priority 3 of defect density (DD_3) for example 4.

4.5.1 Compute the correlation coefficient

Computing the correlation coefficient is the first step in doing simple regression analysis. The correlation coefficient can be used to check if a strong relationship really exists between the dependent and the independent variables. There are many statistical techniques that can be used to calculate the correlation coefficient. Some of the techniques are Pearson's coefficient, Kendall's tau-b, Spearman's correlation. Kendall's tau-b and Spearman's correlation are used to analyze non-normally distributed data set. Pearson's coefficient was used in this thesis because it was assumed that the data sets are normally distributed. Using the formula provided in Chapter 4.3, the Pearson's

coefficient of the relationship between ERD_2 and DD_3 is -0.3907, which identifies the presence of a significant negative correlation. A weak correlation means it is statistically impossible to determine if a relationship really exists between the dependent and the independent variables.

4.5.2 Scatter diagram

Strong correlation coefficient signals that a linear relationship may exist, For non-linear relationship, scatter diagram is the simplest technique to check whether or not a relationship does exist. By looking at the shape of the scatter diagram, one may be able to tell if there is a relationship between dependent and independent variables. If there is a relationship, one also can choose the corresponding function to represent the data points according to the shape of the graph. For example, the represented function can be a logarithmic or a simple linear regression function. For example, Figure 4.3 shows that there is an inverse linear relationship between the threshold 2 external reuse density (ERD_2) and defect density for severity class 3 (DD_3) for example 4.

4.5.3 Linear regression

4.5.3.1 The linear regression equation

After the scatter diagram showed that a linear relationship may exist, the next step is to use a mathematical linear equation, $\boldsymbol{Y = aX+b}$, to represent the data set. In the equation, $Y = aX+b$, X is the independent variable, and Y is the dependent variable. The independent variable X is assumed to contain no measurement errors. The dependent variable Y is assumed to contain all the random errors associated with the measurement. In real life, it is almost impossible to use the linear regression function to predict the exact value of the dependent variable because few data has perfect linear relationships.

There are many ways to calculate the linear regression equation. The simplest and quickest way is to first draw a straight line by hand that best represents the dots in the

scatter diagram. The slope '*a*' and the Y intercept '*b*' can then be calculated based on the straight line.

Although it is simple to produce, a hand-drawn line has several drawbacks. First, it is almost impossible to draw the best line to represent all the dots in the scatter diagram. Secondly, this is not a reproducible process as no two people can draw an identical line in exactly the same place

One of the most popular mathematical techniques to calculate the best regression equation is called the least-square method.

There are many benefits of using the least-square method. First, it is a reproducible process that allows different people to arrive at the same result. Moreover, it can produce a line with a minimum sum of error between the actual *Y* and the predicted value. The slope and the y-intercept of the linear regression function are calculated below.

$$b = \frac{n\sum X_i Y_i - \left(\sum X_i\right)\left(\sum Y_i\right)}{n\left(\sum X_i^2\right) - \left(\sum X_i\right)^2}$$

$$a = \frac{\sum Y_i}{n} - b\left(\frac{\sum X_i}{n}\right)$$

where

b = the estimate of slope of the regression line

a = the estimate of Y intercept of the regression line

X_i = the value of the independent value

Y_i = the value of the dependent value

n = the number of observations

The linear equation for example 4 is shown below in Figure 4.4 and Table 4.1. Table 4.1 shows the values of the *Y* intercept and the slope of the regression line. Figure 4.4

shows a negative linear relationship between two variables, DD_3 and ERD_2. As shown in Figure 4.4, a mathematical equation is good at predicting the trend of the data. However, it is important to know that the equation may be only valid in a certain time frame or a certain condition as shown by example 4 in chapter 4.4.3 and Figure 4.3-Figure 4.4.

	a	*b*
Project 1	-0.037	0.016

Table 4.1: Regression coefficients for ERD_2 vs. DD_3 in example 4

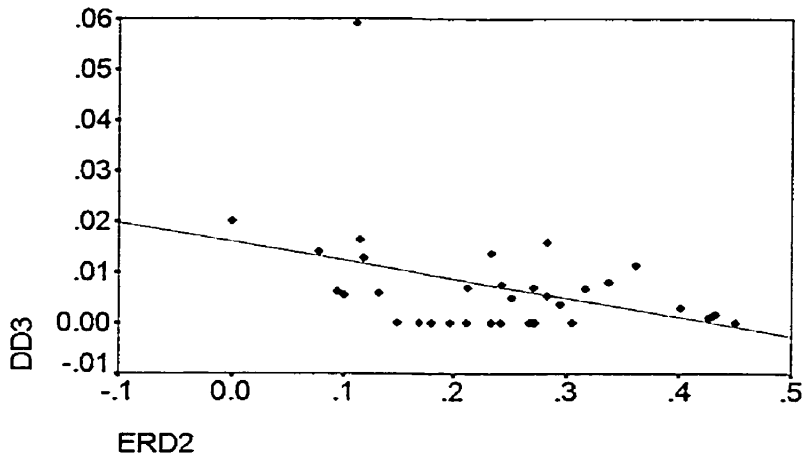

Figure 4.4: linear regression function for ERD_2 vs. DD_3 in example 4

4.5.3.2 Computing the standard error of estimate of dependent variable, Y

In most cases, linear regression function can not fit all the dots in the scatter diagram perfectly. Therefore, error exists between calculated value and actual value. In general, the smaller the error is, the better the line represents the actual data relationship. If the value of error is equal to zero, then the regression line exactly matches the data set. The average value of error can be represented by the standard error of estimate of dependent variable. The formula to calculate the standard error of estimate of dependent variable Y is shown below.

$$S_{yx} = \sqrt{\frac{\sum Y_i - a\sum Y_i - b\sum X_i Y_i}{n-2}}$$

where

S_{yx} = the standard error of estimate of dependent variable Y

Y_i = actual value of the dependent variable

X_i = actual value of the independent variable

n = number of observations

The standard error of estimate of dependent variable represents the average error. The smaller the standard error of estimate is, the better the regression line representing the actual data set is. When the standard error of estimate is equal to zero, all the dots in the scatter diagram lie exactly on the regression line.

4.5.4 Checking the hypotheses

4.5.4.1 Linearity

It is assumed there is a linear relationship existing between X and Y. Linearity can be verified by having significant linear correlation. For example, as shown by the linear correlation in Chapter 4.5.3, there is a linear relationship existing between variables ERD_2 and DD_3 in example 4.

4.5.4.2 Homoscedasticity

Linear regression assumes that the standardized residuals do not change much with each independent value. Homoscedasticity can be checked by plotting the residuals over the mean residual value for each dependent variable hypotheses (Benedicenti *et al.*, 1997), or by visual inspection of the linear regression plot. For example, the analysis of the linear regression plots in Figure 4.5 does not show evidence of any major changes in the patterns of the residues.

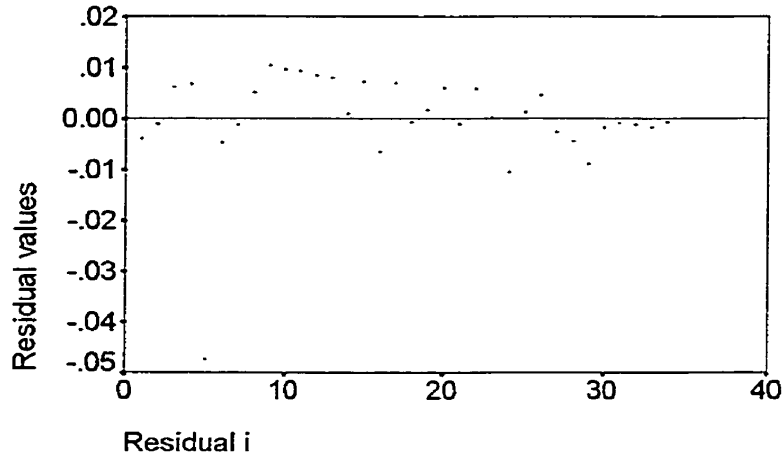

Figure 4.5: Residual plots for ERD_2 vs. DD_3 in example 4

4.5.4.3 Independence of error

The assumption of independence of error is that each error is independent. The error is the difference between the estimated dependent value and the actual value. The independence of error can be verified by using Durbin-Waton statistic, which is a test for serially correlated or autocorrelated residuals (Carlson and Thorne, 1997). The Durbin-Waton statistic measures the correlation of each residual and the residuals proceeding and following it (Benedicenti *et al.*, 1997). When the estimated correlation is zero, the Durbin-Watson statistic is 2. Values close to 0 indicate positive autocorrelation, Values close to 4 indicate negative autocorrelation. The Durbin-Watson value for example 4 is 2.192, which reveals only slight autocorrelation.

4.5.4.4 Normality

It is assumed that the value of the dependent value Y is normally distributed for each value of the independent variable X. Normality can be checked by using a standardized residual plot and by using a standardized residual histogram.

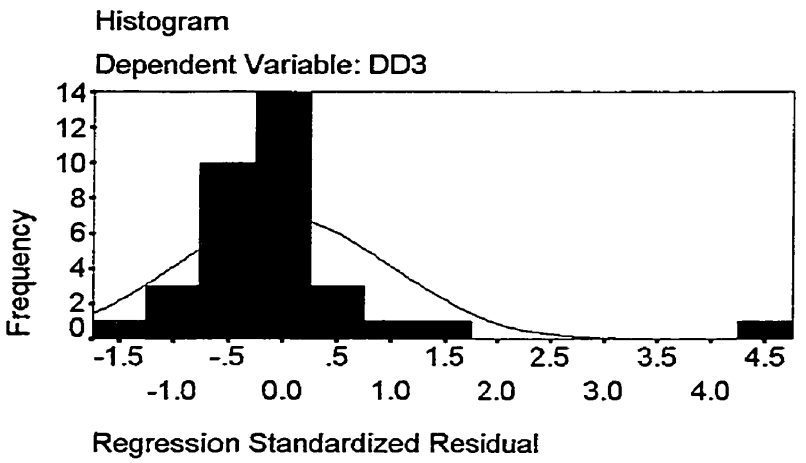

Figure 4.6: Standardized residual histogram for ERD_2 vs. DD_3 in example 4

Except for one extreme value, the histograms in Figure 4.6 show that the standardized residuals in example 4 are approximately normal

By using linearity, normality, homoscedasticity and independence of error, a sound linear model was verified between the dependent variable DD_3 and the independent variable ERD_2 of example 4.

4.6 Summary

Various statistical techniques and their strengths and weaknesses are discussed in this chapter. Furthermore, the reader was presented with a way that uses a combination of several statistical techniques to represent and to analyze the data set:

1. Use simple variable descriptive statistical techniques, such as mean and median, to describe the data set numerically.
2. Use statistical test, such as Pearson's correlation for normally distributed values, to see how the data set represents the whole population.
3. Use various graphical techniques, such as box plot, to improve the data representation.
4. Use correlation and regression analysis to arrive at a mathematical equation to represent the data set and to predict the future trend.

5 Application analysis

5.1 Introduction

The results on an empirical investigation on the effects of software reuse on the severity of software defects from three "C"-based projects are presented in this chapter. The three "C"-based projects were created by a multinational company producing real time telecommunication applications. Names and details of the company and of the timing of the three projects are omitted for confidentiality reasons.

From this empirical investigation, it is hoped to verify that for each project under study the following hypotheses:

1. There is an inverse correlation between amount-of-reuse and defect density.
2. There is an inverse correlation between amount-of-reuse and Defect Density at different severity classes (DD_i).

5.2 Context of the study

The analysis done in this thesis refers to three projects run by a multinational company producing real-time applications for telecommunications. Information follows on the general context of the study, with specific attention to the collected measures, the experimental design, the research hypotheses, the modes of reuse considered, and the data collection process.

5.2.1 The three projects

In the remaining of the thesis, the three projects are referred to as "project 1," "project 2," and "project 3." The vast majority of the features in each project were written in "C". Assembly language was used only for the low-level control of the hardware. Only the "C" portions are analyzed in this thesis because most of the software reuse occurs in such portions and the complete data on the Assembler portions are not available.

Table 5.1 contains information on the projects' components written in C where most of the software reuse occurs. . Details on the development time and effort, the size in lines of code –measured counting the number of semicolons, and the number of files are provided in Table 5.1. Furthermore, the relative time frames of the three projects are provided in Figure 5.1. The specific timing is omitted for confidentiality.

	Project 1	**Project 2**	**Project 3**
Calendar Time	4 years	1 years	2 years
Effort	900 person days	300 person days	400 person days
Size	12,926 LOC	10,014 LOC	16,478 LOC
No. of Files	29	12	34

Table 5.1: Features of the three projects

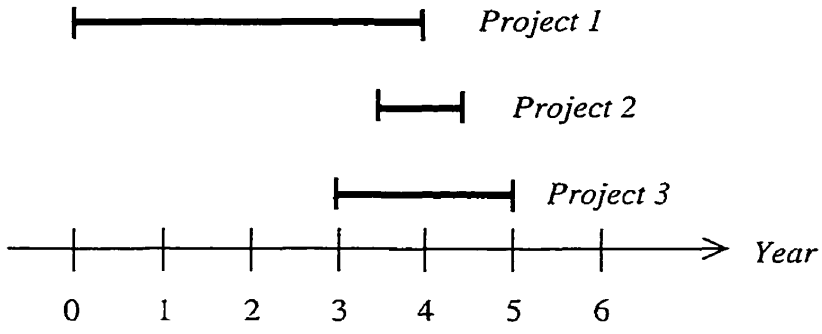

Figure 5.1: Time frames of the three projects

The company organized the development of the three projects in three development teams. The three development teams consisted of a balanced mix of people with at least a BSc in either computer science or electrical engineering, and with up to seven years of programming experience. During the development process, the number of developers in a team ranged from 3 to 5 persons. The assignment of people to teams can be considered as following a random pattern since the teams were roughly equivalent in terms of knowledge, skills and working environment.

All three projects exhibit a low number of lines of code produced per person day, to be expected with the real-time requirements of the target applications. The company

considered the even lower figures of project 1 a natural consequence of the fact that this was the first project undertaken by the firm in the target real-time application domain.

5.2.2 Experimental design

One-short experiment design is used in this thesis (Succi *et al.*, 1998). One-short experiment design is a one-time only collection process. Within each project for each file, the amount-of-reuse metrics (ERL_2, ERF_2, and ERD_2) were compared with defect density and defect density at the various severity levels, (DD_i), (Table 5.2). Other relationships between amount-of-reuse measures and defect measures were also sought. Relations between "conventional" measures (*LOC*, *MCC* and *HV*) and defect measures were sought to determine the original and the new contribution of amount-of-reuse measures to the explanation of defects and defects severity (Table 5.2). The definitions of the metrics were discussed in Chapter 3 in detail.

Independent Variables	Dependent Variables
External Reuse Density$_{threshold2}$ (***ERD$_2$***)	
External Reuse Frequency $_{threshold2}$ (***ERF$_2$***)	Defect density (***DD***)
External Reuse Level $_{threshold2}$ (***ERL$_2$***)	
Lines Of Code (***LOC***)	
McCabe Cyclomatic Complexity (***MCC***)	Defect Density at level *i* (***DD$_i$***)
Halstead Volume (***HV***)	

Table 5.2: Experimental design

5.2.3 The research hypotheses

Because of limited data involved, it is not possible to prove that an actual relationship between amount-of-reuse and software quality actually exist. However, it is statically possible to prove that a relationship is not there. Therefore, this thesis framed statistical questions in terms of null hypotheses. The hypotheses of possible rejections of each project are listed below:

1. There is not a correlation between external reuse density at threshold 2 (ERD_2) and Defect Density (DD) and Defect Density at the different severity classes (DD_i).
2. There is not a correlation between external reuse frequency at threshold 2 (ERF_2) and Defect Density (DD) and Defect Density at the different severity classes (DD_i).
3. There is not a correlation between external reuse level at threshold 2 (ERL_2) and Defect Density (DD) and Defect Density at the different severity classes (DD_i).

As in several studies in software engineering (Melo *et al.*, 1996), the α-level for significance was set at 0.05 in this thesis.

5.2.4 Mode of reuse

As mentioned in Chapter 3.5, the reuse focus of this study is around external code reuse (*ERD*, *ERF* and *ERL*), that is, code reuse coming from outside the scope of the current project (Poulin, 1997).

Developers of the three projects took advantage of an existing company library containing domain specific functions. However, the company did not have a reuse process in place. All the reuse was "*ad-hoc*," based on the decisions of the individual developers to reuse existing functions or procedures instead of creating their own new ones. The content of the reuse library also evolved without control over the duration of the projects. For these reasons, it is not possible to distinguish between the reuse coming from the domain library and the reuse coming from other libraries. The figures on ERF_2, ERD_2 and ERL_2 metrics discussed later in the thesis include both forms of reuse from the domain and from other libraries.

5.2.5 The data collection process

The company employed a product verification team to verify the proper behavior of products. The product verification team was entirely separated from the product development team, and had its own independent process. The product verification team specialized in testing the product before it reached major milestones, such as a beta release, or the final product release. Each time a defect was found, the file containing the defect was identified, a severity was assigned, and the relevant information was entered in a company database.

The defect data have been extracted from the company database after the completion of the work of both the development team and the product verification team. Therefore, the extraction has not created any bias to the usual development process.

The general software measures and the amount-of-reuse measures were collected using WebMetrics tool (Succi *et al.*, 1998), an internal software measurement collection tool.

It would have been interesting to analyze also the actual effort needed to fix each defect. Unfortunately such data were not properly recorded in the database.

5.3 Description and analysis of the results

5.3.1 Descriptive analysis

One of the topics discussed in this thesis is how many significant digits should be choosing in order to represent the values. For this needs error limits. The error limits can be calculated by using bootstrap, which will be discussed in chapter 6. The data in the various analysis tables in the current chapter are currently expressed in four significant digits as typical of many published data. The *dispersion* used in the following tables contains the minimum and the maximum values of the data.

5.3.1.1 Project size analysis

5.3.1.1.1 Descriptive data analysis of project size

Table 5.3 and Table 5.4 below contain the statistic results of the size measures for projects 1, 2, and 3 respectively. From Table 5.3, it can be seen that project 2 has unusually large *means* comparing to project 1 and project 3. Furthermore, project 2 also has larger *standard deviations*. The larger *standard deviations* signal that the *means* may not represent the real data set as well as other projects. The large *means* and *standard deviations* may be contributed by the facts that project 2 has the largest *minimum* values (Table 5.4), and smallest number of files - less than half as many files as project 1 and project 3 (Table 5.1). Moreover, there are very high *kurtosis* for projects 1 and 3 (Table 5.4). The high *kurtosis* means that the data points are more concentrated around the *means*. Finally, projects 1 and 3 also present large positive *skewness* (Table 5.4). A large positive *skewness* means there are more large extreme data then small extreme data in the data set.

	Project1 mean	Project2 mean	Project3 mean	Project1 Std Dev	Project2 Std Dev	Project3 Std Dev	Project1 median	Project2 median	Project3 median
LOC	445.7	834.5	484.7	553.2	667.3	563.8	243.0	596.0	250.0
MCC	123.0	229.9	128.5	154.7	184.5	161.7	56.00	154.00	59.50
HV	29,990	60,270	33,320	41,030	49,530	43,720	13,550	42,690	13,410

Table 5.3: Simple descriptive statistics for project size

	Project1 Dispersion (min - max)	Project2 Dispersion (min - max)	Project3 Dispersion (min - max)	Project1 kurtosis	Project2 kurtosis	Project3 kurtosis	Project1 skewness	Project2 skewness	Project3 skewness
LOC	38 - 2,320	132 - 1,962	38 - 2,320	4.618	-1.243	3.526	2.227	0.558	1.960
MCC	11 - 560	24 - 503	9 - 675	2.492	-1.742	3.576	1.899	0.383	1.982
HV	1,449 - 152,300	7,708 - 139,000	1,469 - 181,600	3.870	-1.311	4.215	2.139	0.549	2.087

Table 5.4: Simple distribution statistics for project size

5.3.1.1.2 Graphical descriptive analysis of project size

The project size histograms and the box-plots for projects 1, 2 and 3 are shown in Figure B.1 to Figure B.6 in appendix. The unusually skewed distributions of the size data for project 1 and project 3 are graphically displayed in the size histograms. The skewnesses of project 1 and project 3 are due to higher percentage of outliers as shown by the box-plots Figure B.4 - Figure B.6.

5.3.1.2 Number-of-defects analysis

5.3.1.2.1 Descriptive data analysis of number-of-defects

Table 5.5 and Table 5.6 below contain the statistic results of the number-of-defects measures for projects 1, 2, and 3 respectively. Table 5.5 shows that project 1 has unusually large number-of-defects by having large *means*, *median* and *maximums*. The larger number-of-defects of project 1 may be due to the fact that project 1 is the first one of a series of projects undertaken by the company. Furthermore, project 1 also has the largest *standard deviations* (Table 5.5) and *dispersion* (Table 5.6) of the three projects. The larger *standard deviation* and *dispersion* show that the *means* in project 1 may not represent the real data set as actually as projects 2 and 3. Moreover, projects 1 and 3 have higher *kurtosis* (Table 5.6). The high *kurtosis* means that the data points are more concentrated around the *means*. Finally, all three projects show a noticeable skewness (Table 5.6).

	Project1 mean	Project2 Mean	Project3 mean	Project1 Std Dev	Project2 Std Dev	Project3 Std Dev	Project1 median	Project2 median	Project3 median
Total # of defects	12.28	4.750	4.850	13.01	4.810	6.460	6	3	2
Total # of S1 Defects	0.552	0.080	0.120	0.985	0.290	0.410	0	0	0
Total # of S2 Defects	3.655	1.420	1.120	5.808	2.070	2.100	2	0	0
Total # of S3 Defects	6.345	2.830	2.650	6.286	1.950	3.634	4	2.5	1.5
Total # of S4 Defects	1.448	0.420	0.940	1.454	0.790	1.410	1	0	0
Total # of S5 Defects	0.207	0.083	0.030	0.491	0.289	0.170	0	0	0

Table 5.5: Simple descriptive statistics for number-of-defects

	Project1 Dispersion (min - max)	Project2 Dispersion (min - max)	Project3 Dispersion (min - max)	Project1 kurtosis	Project2 kurtosis	Project3 kurtosis	Project1 Skewness	Project2 skewness	Project3 Skewness
Total # of defects	2 - 52	0 - 16	0 - 25	2.868	1.446	2.512	1.742	1.421	1.821
Total # of S1 Defects	0 - 4	0 - 1	0 - 2	5.286	12.00	14.54	2.253	3.464	3.752
Total # of S2 Defects	0 - 24	0 - 6	0 - 9	6.937	0.528	6.237	2.602	1.257	2.505
Total # of S3 Defects	0 - 22	0 - 7	0 - 14	0.694	0.531	3.502	1.286	0.723	1.977
Total # of S4 Defects	0 - 5	0 - 2	0 - 4	0.003	1.130	0.394	0.934	1.638	1.343
Total # of S5 Defects	0 - 2	0 - 1	0 - 1	5.738	2.000	34.00	2.442	3.464	5.831

Table 5.6: Simple distribution statistics for number-of-defects

5.3.1.2.2 Graphically descriptive analysis of number-of-defects

The histograms and the box-plots for the number of defects of projects 1, 2 and 3 are in Figure B.7 to Figure B.14 in appendix. The box-plots from Figure B.11 to Figure B.14 show project 1 and project 3 have higher percentage of outliers. The more extreme number-of-defects of projects 1 and 3 maybe due to the fact that projects 1 and 3 have more files as shown in Table 5.1 and the larger *dispersion* as shown Table 5.6.

5.3.1.3 Defect densities analysis

5.3.1.3.1 Data descriptive analysis of defect densities

The number-of-defects alone will not show the whole picture of the quality of the software. Software quality can be looked at from the defect density point of view. Table 5.7 and Table 5.8 below contain the statistic results of the defect density measures for projects 1, 2, and 3 respectively. As shown in Table 5.7, project 1 has unusually high defect density by having large *means* and *maximums*. Furthermore, project 1 also has the largest *standard deviations* of the three projects (Table 5.8). The larger *standard deviation* signals that the *means* in project 1 may not represent the real data set as actually as projects 2 and 3. Finally, projects 1 and 3 have high *kurtosis* and large positive *skewness* (Table 5.8).

	Project1 mean	Project2 mean	Project3 mean	Project1 Std Dev	Project2 Std Dev	Project3 Std Dev	Project1 median	Project2 median	Project3 median
DD	0.035	0.007	0.013	0.026	0.006	0.020	0.029	0.004	0.009
DD_1	0.002	0.000	0.000	0.006	0.000	0.001	0.000	0.000	0.000
DD_2	0.008	0.002	0.003	0.008	0.002	0.010	0.007	0.000	0.000
DD_3	0.019	0.005	0.007	0.018	0.005	0.011	0.014	0.004	0.005
DD_4	0.005	0.001	0.002	0.007	0.001	0.005	0.002	0.000	0.000
DD_5	0.001	0.000	0.000	0.002	0.000	0.002	0.000	0.000	0.000

Table 5.7: Simple descriptive statistics for defect densities

	Project1 Dispersion (min – max)	Project2 Dispersion (min – max)	Project3 Dispersion (min – max)	Project1 kurtosis	Project2 Kurtosis	Project3 kurtosis	Project1 skewness	Project2 skewness	Project3 skewness
DD	0.010 - 0.130	0.000 - 0.020	0.000 - 0.100	6.188	-0.935	12.65	2.190	0.701	3.278
DD_1	0.000 - 0.030	0.000 - 0.000	0.010 - 0.010	17.43	12.00	30.32	3.916	3.464	5.417
DD_2	0.000 - 0.030	0.001 - 0.010	0.000 - 0.060	0.907	2.579	29.21	1.090	1.730	5.247
DD_3	0.000 - 0.070	0.000 - 0.020	0.000 - 0.060	3.428	0.348	16.37	1.898	1.115	3.587
DD_4	0.000 - 0.030	0.001 - 0.003	0.000 - 0.030	6.092	2.856	21.39	2.199	1.919	4.286
DD_5	0.000 - 0.010	0.002 - 0.002	0.000 - 0.010	15.31	12.00	34.00	3.798	3.464	5.831

Table 5.8: Simple distribution statistics for defect densities

5.3.1.3.2 Graphically descriptive analysis of defect densities

The histograms and the box-plots for the defect densities of projects 1, 2 and 3 are shown in Figure B.15 to Figure B.22 in appendix. The histograms show that project 3 has a more skewed distribution. Furthermore, the box-plots from Figure B.19 to Figure B.22 show project 1 and project 3 have higher percentage of outliers.

5.3.1.4 Amount-of-reuse analysis

5.3.1.4.1 Descriptive data analysis of amount-of-reuse

Table 5.9 and Table 5.10 below contain the statistic results of the amount-of-reuse for project 1, project 2, and project 3 respectively. Table 5.9 and Table 5.10 show that all three projects are very similar in terms of software reuse by having similar statistic data. The only major different is projects 1 and 3 are positively skewed (Table 5.10).

	Project1 mean	Project2 Mean	Project3 mean	Project1 Std Dev	Project2 Std Dev	Project3 Std Dev	Project1 median	Project2 median	Project3 median
ERD_2	0.260	0.270	0.240	0.100	0.110	0.110	0.277	0.278	0.244
ERF_2	0.570	0.590	0.540	0.180	0.160	0.200	0.593	0.624	0.591
ERL_2	0.264	0.242	0.238	0.105	0.067	0.110	0.268	0.254	0.247

Table 5.9: Simple descriptive statistics for amount-of-reuse

	Project1 Dispersion (min - max)	Project2 Dispersion (min - max)	Project3 Dispersion (min - max)	Project1 kurtosis	Project2 kurtosis	Project3 kurtosis	Project1 skewness	Project2 skewness	Project3 skewness
ERD_2	0.090 - 0.540	0.050 - 0.430	0.000 - 0.450	0.935	-0.054	-0.531	0.243	-0.486	0.069
ERF_2	0.190 - 0.850	0.180 - 0.780	0.000 - 0.850	-0.117	3.222	0.264	-0.586	-1.457	-0.681
ERL_2	0.061 - 0.490	0.105 - 0.368	0.000 - 0.588	-0.578	0.197	0.314	0.092	-0.616	0.185

Table 5.10: Simple distribution statistics for amount-of-reuse

5.3.1.4.2 Graphically analysis of amount-of-reuse

The histograms and the box-plots for the amount-of-reuse of projects 1, 2 and 3 are shown in Figure B.23 to Figure B.28 in appendix. The histograms show that all three projects have distributions close to normal. Furthermore, there are few extreme data of amount-of-reuse in all three projects as shown by the box-plots from Figure B.26 to Figure B.28.

5.3.2 Overall descriptive analysis

As shown by the statistic tables, the histograms and the box-plots, all values from projects 1 and 3 have skewed distributions, except for the amount-of-reuse. Those extreme values may affect the results of the correlation analysis and the slope and y-intercepts calculations of the linear equation in the later sections.

The numbers of defects in severity 1 and in severity 5 classes are low. The classifications of various severities were provided in Chapter 3.3. The low numbers of defects in severity 1 and severity 5 may be contributed by the following reasons:

- Severity 1: the software systems were designed to deal with critical scenarios; the developers proceeded very carefully, trying to avoid all possible situations where the failure in the software resulted in failures or suspensions of service of the surrounding systems.
- Severity 5: the requirements for the GUIs of the three projects were very simple: the company was concerned about the functionality and the reliability of the resulting systems and not of creating fancy user interfaces.

Given the low number of severity 1 and severity 5 defects, it is not possible to draw any meaningful conclusion about defects in severity 1 and severity 5. Therefore, severity 1 and severity 5 defined defects were not considered in the following analysis.

The analysis proceeds as follows: First are the significance levels of the correlation between amount-of-reuse and defect-density measurement calculations. The significance levels are used to determine whether the variation in defects can be explained by "conventional" software measures. The conventional software measures are classified as the Lines of Code (*LOC*), McCabe Cyclomatic Complexity (*MCC*) (McCabe, 1976) and the Halstead Volume (*HV*) (Halstead, 1977). The three software measurements are defined in Chapter 3. In this way, it is possible to determine the original contribution of measures of reuse to the explanation of Defect Density and Defect Density at the different levels (DD_i). Where a significant correlation exists, a further investigation is proceeded to determine the feasibility to develop a linear model.

5.3.3 Correlation Analysis

Table 5.11, Table 5.12, and Table 5.13 contain the correlations between amount-of-reuse measures and defect measures and between "common" software measures and defect measures. An asterisk, "*", identifies correlations that are significant at the 0.05 level.

	DD_2	DD_3	DD_4	DD
ERD_2	-0.01	-0.22	-0.10	-0.18
ERF_2	0.01	**-0.46***	-0.26	**-0.39***
ERL_2	0.11	**-0.42***	-0.17	-0.28
LOC	0.02	-0.22	-0.22	-0.25
MCC	0.08	-0.22	-0.25	-0.24
HV	0.06	-0.24	-0.26	-0.27

Table 5.11: Project 1 Pearson Correlation results

	DD_2	DD_3	DD_4	DD
ERD_2	0.22	-0.50	0.13	-0.25
ERF_2	0.24	**-0.61***	0.17	-0.31
ERL_2	0.37	-0.32	0.19	-0.06
LOC	0.08	-0.52	-0.01	-0.35
MCC	0.13	-0.50	-0.04	-0.33
HV	0.01	**-0.59***	-0.08	-0.44

Table 5.12: Project 2 Pearson Correlation results

	DD_2	DD_3	DD_4	DD
ERD_2	-0.13	**-0.39***	-0.14	-0.30
ERF_2	-0.20	**-0.44***	-0.06	**-0.35***
ERL_2	-0.27	**-0.35***	-0.07	**-0.32***
LOC	-0.10	-0.13	-0.07	-0.16
MCC	-0.10	-0.14	-0.09	-0.17
HV	-0.09	-0.14	-0.08	-0.17

Table 5.13: Project 3 Pearson Correlation results

The results of the correlation analysis are summarized in Table 5.14. A "yes" identifies the presence of a significant correlation, while a "no" signals the absence of a significant correlation. For easiness of read, the cells containing the "*yes*" of significant correlations have been bolded and grayed.

	Project 1	Project 2	Project 3
ERD_2 vs. DD_2	No	No	No
ERD_2 vs. DD_3	No	No	Yes
ERD_2 vs. DD_4	No	No	No
ERD_2 vs. DD	No	No	No
ERF_2 vs. DD_2	No	No	No
ERF_2 vs. DD_3	**Yes**	**Yes**	**Yes**
ERF_2 vs. DD_4	No	No	No
ERF_2 vs. DD	**Yes**	No	**Yes**
ERL_2 vs. DD_2	No	No	No
ERL_2 vs. DD_3	**Yes**	No	**Yes**
ERL_2 vs. DD_4	No	No	No
ERL_2 vs. DD	No	No	**Yes**
LOC vs. DD_2	No	No	No
LOC vs. DD_3	No	No	No
LOC vs. DD_4	No	No	No
LOC vs. DD	No	No	No
MCC vs. DD_2	No	No	No
MCC vs. DD_3	No	No	No
MCC vs. DD_4	No	No	No
MCC vs. DD	No	No	No
HV vs. DD_2	No	No	No
HV vs. DD_3	No	No	**Yes**
HV vs. DD_4	No	No	No
HV vs. DD	No	No	No

Table 5.14: Summary of the significance of the correlations

Significant correlations are presented only between:

1. External reuse frequency of threshold 2 (ERF_2) and Defect severity 3 density (DD_3) in all the projects.
2. External reuse frequency of threshold 2 (ERF_2) and Defect density (DD) in projects 1 and 3.
3. External reuse level of threshold 3 (ERL_2) and Defect severity 3 density (DD_3) and in projects 1 and 3.
4. External reuse level of threshold 3 (ERL_2) and Defect density (DD) and in projects 3.
5. External reuse density of threshold 2 (ERD_2) and Defect severity 3 density (DD_3) in project 3.
6. Halstead Volume (HV) and Defect severity 3 density (DD_3) in project 3.

It is evident that there is a systematic presence of significant correlations only between ERF_2 and DD_3, which occurs in all three projects. Therefore, the linear regression was done only between the external reuse frequency of threshold 2 (ERF_2) and the defect severity 3 density (DD_3).

As mentioned in Chapter 2, defect density (DD) is calculated as:

$$DD = DD_1 + DD_2 + DD_3 + DD_4 + DD_5$$

Therefore, it is interesting to determine if the correlation between ERF_2 and DD is only a result of the correlation between ERF_2 and DD_3. To do so, we separate the contribution of DD_3 from DD defining $DD\text{-}3$ as:

$$DD\text{-}3 = DD - DD_3 = DD_1 + DD_2 + DD_4 + DD_5$$

Table 5.15 contains the result of the correlation between ERF_2 and $DD\text{-}3$. No significant correlation is present which indicates defect severity 3 errors has significant correlation only with ERF_2.

	Project 1	Project 2	Project 3
ERF_2 vs. DD-3	-0.123	-0.111	-0.121

Table 5.15: Correlation between ERF_2 and *DD-3* in the three projects

5.3.4 Scatter Diagram Analysis

As shown in Figure 5.2, the data points of projects 1, 2 and 3 can be represented by linear regression functions.

5.4 Development of a linear model

The next step in the analysis is to determine a linear model of the kind:

$$DD_3 = a \times ERF_2 + b$$

A univariate linear regression involving ERF_2 as independent variable and DD_3 as dependent variable reveals that in each project there is linear relation between the two variables (Figure 5.2).

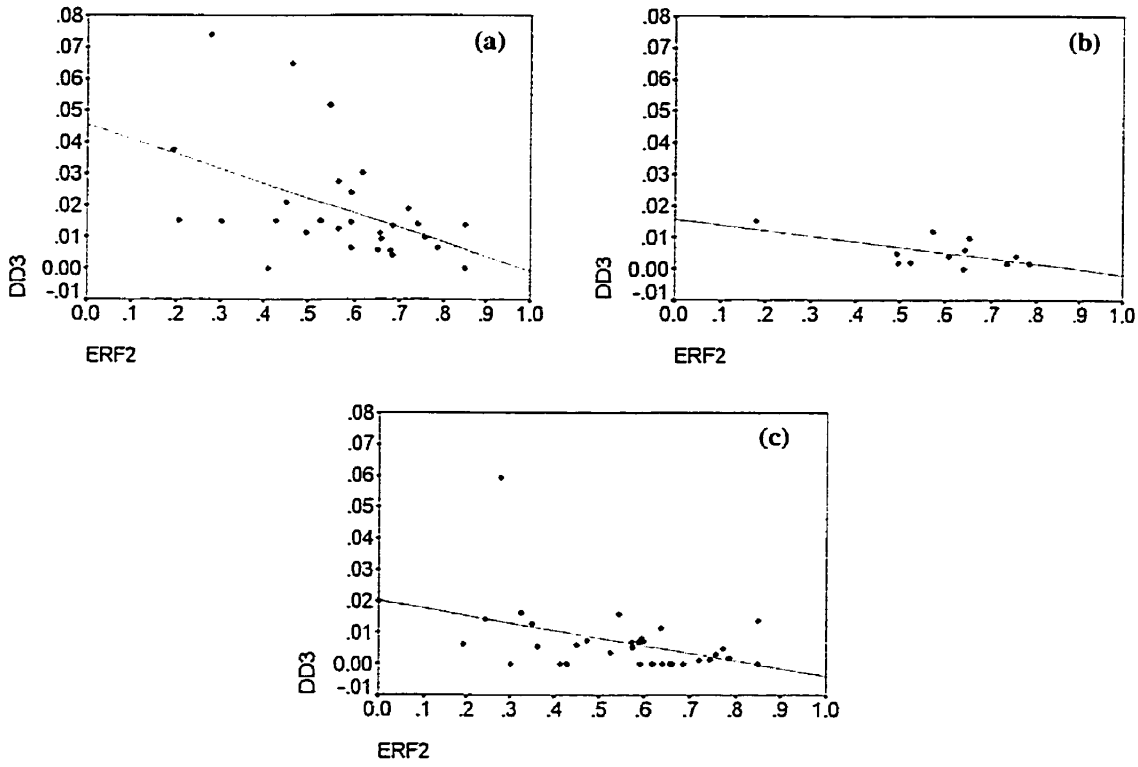

Figure 5.2: Linear regression lines of DD_3 and ERF_2 for project 1 (a), project 2 (b), and project 3 (c)

Table 5.16 contains the regression coefficients for the three projects. All the six coefficients are significant at the 0.05 level.

	a	*b*	St. Error
Project 1	-0.047	0.046	0.016
Project 2	-0.017	0.015	0.004
Project 3	-0.023	0.019	0.010

Table 5.16: Regression coefficients for the three projects

5.4.1 Hypothesis verification

To have a sound linear model, it is necessary to verify linearity, normality, homoscedasticity and independence of error (Benedicenti *et al.*, 1997). The following hypothesis verifies the linear relationships between external reuse frequency at threshold 2 (ERF_2) and priority 3 of defect density (DD_3).

- Linearity: There is a linear relationship existed between variables ERF_2 and DD_3 in each project.
- Normality: The distributions of the residuals of each project in Figure 5.3 are approximately normal.

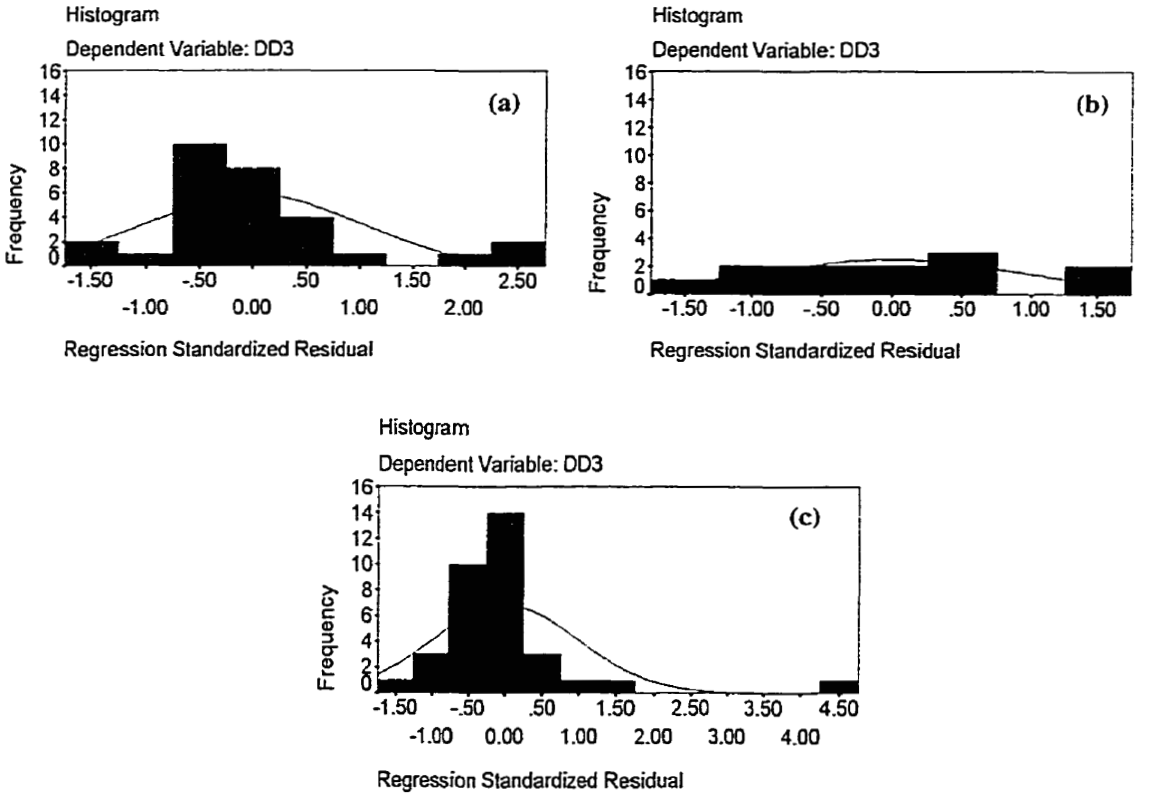

Figure 5.3: Standardized residual histogram for project 1 (a), project 2 (b), and project 3 (c)

- Homoscedasticity: The analysis of the linear regression plots in Figure 5.4 does not show evidence of any major changes in the patterns of the residues. The lines in the middle of the residual plots in Figure 5.4 are the means of the residues.

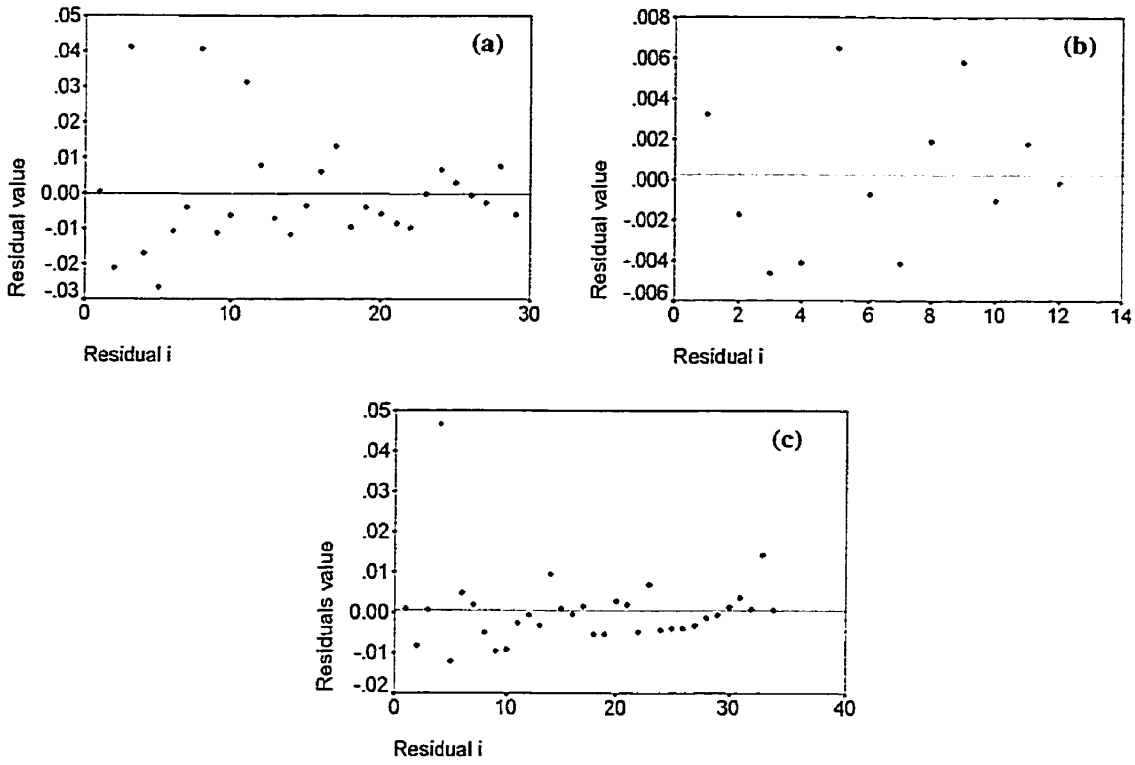

Figure 5.4: Residual plots for project 1 (a), project 2 (b), and project 3 (c)

- Independence of error: The Durbin-Watson coefficients in Table 5.17 reveal only slight autocorrelation in project 1 and 3. However, project 2 does exhibit strong positive autocorrelation which means the residuals for consecutive observations in project 2 may be correlated.

	PROJECT 1	PROJECT 2	PROJECT 3
ERF_2 VS. DD_3	**2.46**	**1.01**	**2.18**

Table 5.17: Durbin-Watson coefficient

The Hypothesis verification shows that there are grounds to support linear models in Project 1 and 3, while the situation for Project 2 is unclear.

5.5 Discussion of the results

5.5.1 Internal validity:

To avoid any bias introduced by the data collection process, the data was analyzed only after the data collection was done by an independent product verification group. Thus, the designers and the testers are not aware of the data analysis plan. Furthermore, to avoid bias, the product verification group was made up by a team of well train testers who verified only the finished programs from the programmers. The defects were identified and recorded according to the company's guidelines. The assignment of the group followed what can be considered a random pattern: the team compositions were equivalent, consisting of both experienced and inexperienced programmers according to the standard company patterns. Altogether, these facts have ensured the overall internal validity of the study.

5.5.2 External validity – Generalization of the results:

The results described in this thesis are significant at the 0.05 level. The results of projects 1, 2, and 3 are:

- There is no correlation between amount-of-reuse (ERD_2, ERF_2, ERL_2) and defect density (DD).
- There is a linear relationship only between external reuse frequency at threshold 2 (ERF_2) and Defect Density of priority 3 errors (DD_3).

The priority 3 error is defined in Chapter 3.3 as part of the system feature does not work as specified. The conclusions resulted from the statistical analysis process can not by itself extended to the entire population in general, but to projects that have one or more similar parameters. Namely:

- Real time C language projects
- Business oriented application
- Project with tracking mechanism
- Defect classifications by severity

5.6 Conclusion

An analysis of three real-time C projects in the telecommunication domain from a North American company has no strong evidence that amount of reuse significantly negatively correlates with defect densities of every severity level. However, it seems that one relationship does exist between amount of reuse and software defect density at severity class i, (ERF_2 and DD_3).

The generalizability of the results is unclear. It is possible that the results are extensible to a larger population of real-time systems, however, more data are required. The results of the data analysis will be discussed more in chapter 7.

6 Bootstrapping

6.1 Introduction

Basic statistical techniques work best when there is a large amount of data. However, there are many situations in which it is impossible or too expensive to collect a large amount of data. In such a case, a technique called bootstrap can be used to analyze the small amount of data in which the conventional statistic is not valid (Zoubir and Boashash, 1998). Bootstrap technique is used in this chapter to improve the accuracy of the results discussed in the previous chapter using conventional techniques.

First introduced are the bootstrap technique used in this thesis, namely: bootstrap mean of correlation calculation, standard deviation calculation, confidence interval calculation and linear regression analysis using bootstrap technique. Next, there is a brief introduction to jackknife technique, which is similar in principle to bootstrap. Finally, the results of using the bootstrap technique are presented and discussed.

6.2 Bootstrap principle

Efron first introduced the bootstrap as a new way to estimate standard error (Efron and Tibshirani, 1993). When there is not enough data to provide a high confidence level, an analyzer would often repeat the data collection process multiple time to improve his confidence. Bootstrap applies the same principle in the way of computer calculation (Efron and Tibshirani, 1986). Instead of repeating the collection process, bootstrap creates multiple data sets; each containing randomly reassigned data from the original sample. As a result, the basic principle of bootstrap is to randomize data so the result will have a Monte Carlo distribution (random distribution). The bootstrap estimation will be closer to the actual population as the number of re-samples or bootstrap replicate data sets approach infinitely large. However, an infinite resample size is impractical. Since the choice of the re-sample size directly affects the estimation accuracy, it is important to choose the right re-sample size. The choice of re-sample size is highly dependent on the

type of data being estimated. Usually, a re-sample size of 100 is sufficient for estimating the standard of error, and a sample size of 1000 is needed for calculating a confidence interval since the confidence interval is a more ambitious measurement (Efron B, 1979).

6.2.1 Bootstrap Correlation's distribution

Efron and Tibshirani (1993) show two ways of calculating regression correlation, pairs-method and residuals-methods. Pairs-method is used in this thesis because it is less sensitive to assumptions than residuals-method (Efron and Tibshirani, 1993).

Example 5: A basic bootstrap correlation distribution calculation between external reuse frequency at threshold 2 (ERF_2) and priority 3 of defect density (DD_3) of project 1 is shown below:

1. Use ERF_2 and DD_3 of project 1 as a sample data set where ERF_2={0.565, 0.593 ... 0.652} and where DD_3={0.027, 0.024 ... 0.006}, each consists of 34 observed data and have a correlation of -0.460.
2. Randomly select a pair of data, ERF_{2i}* from ERF_2 which in this case, ERF_{2i}* = 0.683, and DD_{3i}* from DD_3 which in this case, DD_{3i}* = 0.004. Note, ERF_{2i}* and DD_{3i}* are being selected from ERF_2 and DD_3 with replacement.
3. Repeat step 2, *34* times to create a bootstrap re-sample, which in this case is ERF_2*={0.683, 0.303 ... 0.546} and DD_3* ={0.004, 0.015 ... 0.052} consisting of the same size of random data from both ERF_2 and DD_3. The re-sample size 34 is the total size of the observed data.
4. Calculate a new bootstrap correlation U* between ERF_2 and DD_3 based on the data sets from step 3.
5. Repeat step 2, 3, 4, m times to get U_1*,..., U_m*. Note, m should be a large number, usually greater than 30. In this example, m is 1000.
6. The U_1*,..., U_m* is the bootstrap's estimated correlation distribution between ERF_2 and DD_3.

A bootstrap mean would be similar to the actual mean in a random data set. A vast different bootstrap mean with its actual mean may signal an abnormal data distribution. The bootstrap correlation distribution mean between ERF_2 and DD_3 is the mean of the data distribution set in example 5's step 6. The bootstrap correlation distribution mean is -0.458 which is close to the actual correlation, -0.460, as calculated in the previous chapter.

6.2.2 Standard deviation using bootstrap

The steps for calculating standard deviation using bootstrap are identical to the steps shown in example 5. The only difference is in step 6. Instead of estimating the mean, the standard deviation is calculated by using the following formula:

$$\sigma = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}\left(\upsilon^*_i - m^{-1}\sum_{i=1}^{m}\upsilon^*_i\right)^2}$$

where: m = re-sample times which is 1000 in example 5

υ^*_i = re-sample item which in this case is ERF_2* in example 5

σ = standard deviation

At lease two significant digits are need to separate one result from another based on the bootstrap data analysis done in Table 6.2. Therefore, all the data are going to be represented in two significant data in the following sections. Based on the data from example 5, the bootstrap standard deviation is 0.19.

6.2.3 Bootstrap confidence interval

Zoubir and Boashash (1998) showed several ways of using the bootstrap to calculate confidence interval and hypothesis. Example 6: Confidence interval for mean of correlation using bootstrap is shown below.

1. Use ERF_2 and DD_3 of project 1 as a sample data set where ERF_2={0.565, 0.593 ... 0.652} and where DD_3={0.027, 0.024 ... 0.006}, each consists of 34 observed data and have a correlation of -0.46.
2. Randomly select a pair of data, ERF_{2i}* from ERF_2 which in this case, ERF_{2i}* = 0.683, and DD_{3i}* from DD_3 which in this case, DD_{3i}* = 0.004. Note, ERF_{2i}* and DD_{3i}* are being selected from ERF_2 and DD_3 with replacement.
3. Repeat step 2, 34 times to create a bootstrap re-sample, which in this case is ERF_2*={0.683, 0.303 ... 0.546} and DD_3* ={0.004, 0.015 ... 0.052} consisting of the same size of random data from both ERF_2 and DD_3. The re-sample size 34 is the total size of the observed data.
4. Calculate a new bootstrap correlation U* between ERF_2 and DD_3 based on the data sets from step 3.
5. Repeat step 2, 3, 4, m times to get U_1*,..., U_m*. Note, m should be a large number, usually greater than 30. In this example, m is 1000.
6. Sort the bootstrap estimates in ascending order such that U_1* < U_2* < ...< U_{1000}*. For example, the sorted list may look like -0.87, -0.84,...., -0.033.
7. Calculate the confidence interval from the sorted list in step 6. The confidence interval is (ERF_{c1}, ERF_{c2}), where $c1 = ma / 2$, and $c2 = m - c1 + 1$. 'm' is the number of the estimation, and 'a' is the confidence level. For example, when a = 0.05, and m = 1000, we get cl_{25} =-0.70, and cl_{976} = -0.19.

The actual statistic correlation is -0.46, and bootstrap standard deviation is 0.19. The range of the correlation, –0.46±0.19, is within the bootstrap confidence interval of (-0.70, -0.19). It is 95% confidence to say that the correlation between ERF_2 and DD_3 is – 0.46±0.19.

6.2.4 Bootstrap's Linear Regression analysis (Slope & y-intercept)

Zoubir and Boashash (1998) showed several ways of using the bootstrap to do linear regression analysis. Example 7 for using bootstrap to calculate slope and y-intercept is shown below.

1. Use ERF_2 and DD_3 of project 1 as a sample data set where ERF_2={0.565, 0.593 ... 0.652} and where DD_3={0.027, 0.024 ... 0.006}, each consists of n = 34 observed data and have a slope of -0.047 and y-intercept of 0.046.
2. Randomly select a pair of data, ERF_{2i}* from ERF_2 which in this case, ERF_{2i}* = 0.683, and DD_{3i}* from DD_3 which in this case, DD_{3i}* = 0.004. Note, ERF_{2i}* and DD_{3i}* are being selected from ERF_2 and DD_3 with replacement.
3. Repeat step 2, 34 times to create a bootstrap re-sample, which in this case is ERF_2*={0.683, 0.303 ... 0.546} and DD_3* ={0.004, 0.015 ... 0.052} consisting of the same size of random data from both ERF_2 and DD_3. The re-sample size 34 is the total size of the observed data.
4. Using the formulas below to calculate a new bootstrap slope (S*) and y-intercept (y*) using the data from ERF_2* and DD_3* from step 3 (Cangelosi *et al.*, 1983).

$$slope = \frac{n\sum XY - \left(\sum X\right)\left(\sum Y\right)}{n\left(\sum X^2\right) - \left(\sum X\right)^2}$$

$$y - intercept = \frac{\sum Y}{slope} - b\left(\frac{\sum X}{slope}\right)$$

5. Repeat step 2, 3, 4, m times to get S_1*,..., S_m* and get y_1*,..., y_m*. Note, m should be a large number, usually greater than 30. In this example, m is 1000.
6. Calculate the standard deviation and the 95% confidence interval of the slope and y-intercept based on the list from step 5. The standard deviations in this example are, slope=0.020 and y-intercept=0.014. The confidence interval in this example are, slope=(-0.093,-0.010) and y-intercept=(0.021,0.077).

6.2.5 Jackknife technique

One of the weaknesses in using the bootstrap technique is that there is no limit on how large a random sample should be. Therefore, one may need to try several random sample sizes in order to find the one that produces the result with the needed actuality.

Besides bootstrap, there is another technique called jackknife that always uses a fixed number of sampling. The jackknife can be thought of as a linear approximation to the bootstrap (Efron and Tibshirani, 1993). The steps in doing jackknife are almost identical to doing bootstrap. The only difference is that jackknife draws n samples of size n-1 each time without replacement from the original sample of size n (Zoubir and Boashash, 1998).

Unlike the bootstrap formulas for calculating standard deviation in Chapter 6.2.2, The jackknife for estimating the standard deviation is:

$$\sigma = \sqrt{\frac{m-1}{m}\sum_{i=1}^{m}\left(\upsilon_i - m^{-1}\sum_{i=1}^{m}\upsilon_i\right)^2}$$

where: m = re-sample times

υ^*_i = re-sample item

σ = standard deviation

As shown above, jackknife has fixed re-sampling times, m. Thus, the analyzer using jackknife technique does not have to make an educated guess on the re-sample times as in the bootstrap case. However, jackknife method would be useless if the data can not be approximated linearly (Efron B, 1979). Furthermore, there is some estimating efficiency lost in using jackknife (Efron and Tibshirani, 1986). As a result, bootstrap technique is used in this thesis to calculate the means, standard deviations and confidence intervals.

6.3 Bootstrap Analysis

Table 6.1 provides a summary on various bootstrapping techniques that will be used in this thesis.

Bootstrap type	Definition
Bootstrap Correlation Mean (Efron and Tibshirani, 1993)	The mean of the '*m*' correlations of '*n*' randomly selected data with replacement of the same original size *n*.
Bootstrap Std. Deviation (Zoubir and Boashash, 1998)	The standard deviation of the '*m*' correlations of '*n*' randomly selected data with replacement of the same original size *n*.
Bootstrap Confidence Interval (Zoubir and Boashash, 1998)	The 95% confidence interval of the '*m*' correlations of '*n*' randomly selected data with replacement of the same original size *n*.

Table 6.1: Bootstrapping techniques

Table 6.2 - Table 6.3 below contains the comparisons between bootstrapping data and the actual statistic data. Furthermore, the tables below also list the results of standard deviation and 95% confidence interval calculations for correlation analysis and linear regression analysis done on ERF_2 and DD_3.

	Project 1	Project 2	Project 3
Actual correlation	-0.46	-0.61	-0.44
Bootstrap correlation mean	-0.46	-0.50	-0.45
Bootstrap Std. Deviation of the correlation	0.19	0.20	0.13
Upper 95% correlation confidence interval	-0.19	0.21	-0.15
Lower 95% correlation confidence interval	-0.70	-0.88	-0.71

Table 6.2: Bootstrapping correlation data vs. actual correlation data

	Project 1	Project 2	Project 3
Actual y-intercept	0.046	0.015	0.019
Bootstrap y-intercept	0.046	0.014	0.020
Std Dev of the y-intercept	0.014	0.006	0.008
Upper 95% y-intercept confidence interval	0.077	0.02	0.038
Lower 95% y-intercept confidence interval	0.021	0.0004	0.007

Table 6.3: Bootstrap's y-intercept data vs. actual y-intercept data

	Project 1	Project 2	Project 3
Actual slope	-0.047	-0.017	-0.023
Bootstrap slope	-0.047	-0.015	-0.024
Std Dev of the slope	0.020	0.009	0.012
Upper 95% slope confidence interval	-0.010	0.0059	-0.004
Lower 95% slope confidence interval	-0.093	-0.025	-0.051

Table 6.4: Bootstrap's slope data vs. actual slope data

As shown in Table 6.2 - Table 6.4, the ranges for the 95% confidence interval are all very large. Therefore, even though the confidence intervals for project 1, 2 and 3 overlap that alone still can not prove the statistic results on the three projects are related. However, the values, actual correlation ± standard deviation, of the three projects are all within their 95% confidence intervals respectively. Thus further strength the conclusion from the previous chapter that there is a strong negative correlation between ERF_2 and DD_3. The large difference between the actual correlation and the bootstrap correlation mean can be explained by the fact the project 2 has very few data points, only 12 data points. Thus, the extreme data points have a large effect on the actual correlation calculation. Unlike the correlation data in Table 6.2, the bootstrap data of the three projects, y-intercept and slope, in Table 6.3 and Table 6.4 are all larger than their standard deviation, which could be interpreted as that the amount of reuse may affect defect density differently in the three projects.

6.4 Summary

The results from the bootstrap analysis show that there is a negative relationship between external reuse frequency at threshold 2 (ERF_2) and priority 3 of defect density (DD_3) in all three projects. However, the effects of amount of reuse, ERF_2, on defect density at severity class 3, DD_3, in three projects may not be the same because the linear relationship of one project is not identical to the linear relationship of anther project as shown in the bootstrap analysis.

Theoretical and practical work have shown that bootstrap methods are potentially superior to large-sample technique (Zoubir and Boashash, 1998). However, care should still be given in using the bootstrap. Like any other estimation techniques, its accuracy is also greatly dependent on the sample set. Furthermore, the bootstrap method should be used as an alternative, rather than as a replacement for the standard statistical procedures.

7 Conclusions

7.1 The work done in this thesis

Investigate is done in this thesis to see whether or not software reuse decreases the Defect Density of different severity levels, DD_i, in three real-time telecommunication projects. In doing so, I not only need to use various statistic techniques, but also have to develop programs to calculate correlation, and to do bootstrapping.

Various software reuse and software quality metrics are presented in the beginning of this thesis as background information. After that, five software attributes used in the metrics calculation in the analysis section of the thesis are defined in Chapter 3. The five software attributes are *lines-of-code*, *Halstead's Volume*, *defect severity*, *McCabe cyclomatic complexity of a program* and *amount-of-reuse measures*. After the discussion of the metrics and their atrributes, various statistic techniques used to discribe and to analyze the data sets of this thesis are then introduced. Once the background information is introduced, empirical investigations are done on the three-telecommunication projects using previous introduced statistical techniques. The statistical techniques contain descriptive techniques, graphical representations, and correlation and regression analysis. Finally, bootstrapping techniques are used to improve the analysis results using normal statistical techniques.

7.2 The conclusions drawn from the analysis

Software reuse benefits are found in many published papers. In general, the benefits are lower development cost (Thomas *et al.*, 1995), increased productivity (Gaffney and Cruickshank 1992), and lower defect density (Succi and Benedicenti, 1998). Significant errors and defect density decrease as a direct result of software reuse are reported in many software development stages and in many publications as shown below:

- Agresti and Evanco (1992) saw a positive Log-Linear relationship between FNEMC (fraction of total compilation units that are new or extensively modified) and higher defect density.
- Melo *et al.* (1995) saw that higher reuse rate decreases the error density.
- (Basili *et al.*, 1996) found that higher reuse rates correlates with lower defect density.
- Devanbu *et al.* (1996) concluded that a negative Log-linear relationship existed between reuse size and error density, and between reuse ratio and error density.
- Frakes and Succi (1997) reported that higher reuse level and reuse frequency correlates with higher quality ratings and lower levels of defects and deltas.
- Succi and Benedicenti (1998) saw higher reuse density corresponding to lower customer complaint density.

Unlike most published software reuse results, significant correlations between amount of reuse (reuse level, reuse frequency and reuse density) and Defect Density at different severity levels are not found in this thesis. The only exception is a strong correlation shown between reuse frequency and Defect Density at the third severity level (DD_3). The third severity level error is defined in Chapter 3.3 as part of the system feature does not work as specified. Due to the small number of files involved, 75 files in 3 projects, it is not possible to draw a general conclusion that software reuse has no effects on software quality in terms of Defect Density at level *i*. However, the results of this thesis do point out that using software reuse will not guarantee immediate software quality improvement in every software project.

There are many factors that may contribute to the discrepancy between the results of this thesis and other papers that reported various software reuse benefits. One of factors maybe because people usually do not report an undetermined relationship. Therefore, the papers that have published are the projects that showed a strong negative relationship between software reuse and software quality. However, a few reports did mention an undetermined or positive relationship between software reuse and software quality while

at the same time showing the other software reuse benefits, i.e., lower development cost. One of such paper is by Devanbu *et al.*, (1996) who reported a positive relationship between reuse level and error density.

Another reason, that may contribute to the negative showing of a strong correlation between amount of reuse and software quality in this thesis, may be due to the fact the company does not have a software reuse process. Poulin (1997) states that such "*ad-hoc*" type reuses, copying and modifying software or component, have limited benefits. In the "*ad-hoc*" type reuse, the reused components have to go through the same testing, and documentation as the new components.

7.3 Recommendations for future works

Some suggestions for the continuation of the work of this thesis are to analyze what the type of software process and what kind of projects that will show a strong negative correlation between amount of software reuse and software Defect Density at the different severity classes (DD_i):

- Analyze the effects of language on software reuse: "C" does not have as many features to facilitate software reuse as the fourth generation languages; such as Java, or as the third generation languages; such as Ada. Therefore, it would be interest to compare the results from the projects written in "C" with the projects written in the newer generation of languages to see how much effects do the type of language have on Defect Density at the different severity classes (DD_i).
- Analyze the effects of software process on software reuse: A company with a software reuse process will save more money in the long term than a company using "*ad-hoc*" software reuse (Poulin, 1997). By comparing projects produced under software reuse process with projects written under "*ad-hoc*" process, the effects of amount of reuse on software Defect Density at the different severity classes under different software processes will be highlighted.

- Analyze the effects of project environment on software reuse: Different types of component take different amount of amount of effort to be reused (Poulin, 1997). Therefore, it is fair to assure different types of project also take different amount of effort to be reused. For example, software reuse may produce fewer benefits in a telecommunication environment than in a window environment.
- Bootstrap use:
  - The bootstrap correlation mean of project 2 in Table 6.2 agreed with project 1 and 3's bootstrap correlation means, but its actual correlation mean did not agree with the results from project 1 and 3. Therefore, which result should be used to represent the data set?
  - By removing the influence of the extreme values in the data set, bootstrap correlation mean of project 2 looks similar to the correlation means of other projects in Table 6.2. What would happen to the results in other papers when bootstrap is used, and the extreme results are not rejected? Also, what would the Pearson correlation results be, if bootstrap were used in Table 5.11 - Table 5.13?
  - Take into consideration when the bootstrap confidence limits are wide.

8 References

1. Agresti, W.W., and W. M. Evanco, "Projecting Software Defects From Analyzing Ada Designs," *IEEE Transactions on software engineering*, Vol. 18, No. 11, pp. 988-997, Nov 1992.
2. Aron A., E.N. Aron, "*Statistics for the Behavioral and Social Sciences,*" Prentice Hall, 1997.
3. Basili, V., L. Briand., and W.Melo, "How Reuse Influences Productivity in Object-Oriented Systems," *Communication of the ACM*, 39(10), Oct 1996.
4. Benedicenti, L., G. Succi., and T. Vernazza, "Guidelines to Determine the Impact of Code reuse on Productivity," *University of Genova*, DIST-LIPS-TR-97002, Mar 1997.
5. Briand, L., K. El Emam, S. Morasca "On the Application of Measurement Theory in Software Engineering," *Empirical Software Engineering*, 1(1), 1996.
6. Bryant, Edware C., "Statistical analysis," Second edition, McGraw-Hill Book Company, 1966.
7. Cangelosi E.V., P.H. Taylor., and P.F. Rice, "Basic Statistics: A Real World Approach," 3rd edition, West Publishing Company, 1983.
8. Card, D.N., and W. W. Agresti, "measuring software design complexity," *J.Syst.Software* Vol 8, pp. 185-197, Mar 1988.
9. Carlson W.L., and B. Thorne, "Applied Statistical Methods for Business, Economics, and the Social sciences," Prentice Hall, 1997.
10. Chen, Y., B. Krishnamurthy, K. Vo, "An Objective Reuse Metric: Model and Methodology," *Fifth European software engineering Conference*, 1995.
11. Cochran, W.G. "Some Consequences when the Assumptions for the Analysis of Variance Are Not Satisfied," *Biometrics*, 3, 1947.
12. Conte, S., H. Dunsmore, V. Shen, "*Software Engineering Metrics and Models,*" Benjamin/Cummings Publishing Co, 1986.
13. Department of Defense Program Manager's Reuse Issues Handbook, Feb 19, 1996.

14. Devanbu, P., S.Karstu., W. Melo, and W. Thomas, "Analytical and Empirical evaluation of Software Reuse Metrics," 18th International Conference on Software Engineering, 1996.

15. Efron B, "Bootstrap methods: Another look at the Jackknife," *The 1977 Rietz lecture*, The Annals of Statistics, Vol 7, No 1, 1-26, 1979.

16. Efron, B and R.J Tibshirani, "Bootstrap Methods for Standard Errors, Confidence Intervals, and Other Measures of Statistical Accuracy," Statistical Science, Vol 1, No 1, 54-77, 1986.

17. Efron, B and R.J Tibshirani, "An Introduction to the Bootstrap," Chapman & Hall Inc., 1993.

18. Fenton, N.E., and Shari Lawrence Pfleeger, "Software Metrics: A rigorous & Practical Approach," Second Edtion, PWS Publishing Company, 1996.

19. Ferri, R., R. Pratiwadi., L. Rivera., M. Shakir., J. Snyder., D. Thomas., Y. Chen., G. Fowler., B. Krishnamurthy., and K. Vo, "Software Reuse Metrics for an Industrial Project," *Proceedings of the 4th International Symposium of software Metrics*, 1997.

20. Frakes, B., and G. Succi, "An Empirical Study of Reuse, Quality, and Productivity," 1997.

21. Frake, W., and C. Terry, "Reuse Level Metrics," *Proceedings 3rd International conference on Software Reuse*, 1994.

22. Frakes, W., and C. Terry, "Software reuse: Metric and Models," *ACM Computing Surveys*, Vol. 28, No. 2, pp. 415-435, June 1996.

23. Gaffney, J.E., and R.D. Cruickshank, "A general Economics Model of Software Reuse," *Proceedings of the 14th International Conference on Software Engineering*, 1992.

24. Hamburg, M., "Statistical analysis for decision making," Harcourt, Brace & World, Inc., 1970.

25. Humphrey, W.S., "Managing the Software Process," Addison Wesley, 1990.

26. Kichenham, B, "Measuring software quality," *First annual software quality workshop*, Aug. 1989.

27. Lion, J.L., "ARIANE 5 Flight 501 Failure, Report by the Inquiry Board," Paris, 19 July 1996.

28. McCabe, T., "A Complexity Measure," *IEEE Transactions on Software Engineering*, 2(4), 1976.

29. Melo, W.L., L.C. Briand., and V. R. Basili, "Measuring the impact of Reuse on Quality and Productivity in Object-Oriented System," *Technical report*, Univ. of Maryland, Dep. Of Computer science, College Park, MD, USA 20742., pp. 1-16, Jan 1995.

30. Merrill, W.C., and K.A. Fox, "Introduction to Economic Statistics," John Welley & Sons Inc, 1970.

31. Mills, Everald E, "Software metrics," *SEI curriculum module* SEI-CM-12-1.1, Dec 1988.

32. Pazer, Harold L., and Lloyd A. Swanson, "Modern Methods for statistical analysis," pp358-366.

33. Pearsons, Robert, "Statistics For Decision Making," Harper & Row Publishers, 1974.

34. Poulin, J. S "Measuring software reuse, principles, practices, and economic models," Addison-Wesley, 1997.

35. Poulin, J., J. Caruso, and D. Hancock, "The business Case for Software Reuse," *IBM System Journal* 32(4), 1993.

36. Sadahiro I., "Experience Report on Software Reuse Project: Its Structure, Activities, and Statistical Results," ACM 0-89791-504-6/ 92/ 0500-0320 1.50., pp. 320-326, 1992.

37. Selby, R.W., and V.R.Basili., "Analyzing error-prone system structure," *IEEE Trans. software Eng.*, vol.17, pp. 141-152, Feb 1991.

38. Succi, G., L. Benedicenti, C. Bonamico, T. Vernazza, "The Webmetrics Project - Exploiting 'Software Tools on Demand'," *Proceedings of the 1998 World*, 1998. *Multiconference on Systemics, Cybernetics, and Informatics*, Orlando, FL, 1998.

39. Succi, G., L. Benedicenti, T. Vernazza, "Analysis of the effects of software reuse on customer satisfaction in an RPG Environment," *Submitted for publication,* 1999.

40. Thomas, William M., Alex Delis., and Victor R. Basili, "An analysis of errors in reuse-oriented development environment," 1995.

41. Tracz, Will, "Software reuse myths revisited," IEEE 0270-5257, 1994.

42. Zoubir A.M., and B. Boashash, "The Bootstrap and its Application in Signal Processing," *IEEE signal processing magazine* 1053-5888/98/$10.00., pp.56-76 Jan 1998.

9 Appendix: A

.000	.077	.094	.100	.111	.115	.117	.131	.148	.168
.180	.197	.212	.213	.234	.234	.243	.244	.252	.268
.272	.272	.283	.283	.294	.306	.317	.337	.361	.401
.426	.429	.432	.449						

Table A.1: Original data

ERD2	.361	.337	.317	.244	.168	.429	.117	.213	.115	.131
DD3	.011	.008	.007	.008	.000	.001	.012	.007	.016	.006
ERD2	.000	.252	.077	.272	.197	.401	.283	.212	.432	.268
DD3	.020	.005	.014	.000	.000	.003	.016	.000	.002	.000
ERD2	.294	.180	.094	.272	.111	.148	.243	.449	.426	.234
DD3	.004	.000	.006	.007	.059	.000	.000	.000	.001	.000
ERD2	.100	.283	.306	.234						
DD3	.006	.005	.000	.014						

Table A.2: Reuse data

B. Appendix

Appendix B contains the graphical representations of the data set of the three projects analyzed in chapter 5. The graphical representations contained in this appendix consisted of histograms and box plots. The diagrams are used to graphically display the project size, number-of-defects, defect-density and amount-of-reuse data. The techniques of using histogram and box plot were introduced in chapter 4, while the interpretations of the graphical representations shown below are in chapter 5. A brief summery of the figures is shown in Table B.1 below.

Statistics types	Name of the figures	Contents of the three projects figures
Project size	Figure B.1	LOC histograms
	Figure B.2	MCC histograms
	Figure B.3	HV histograms
	Figure B.4	LOC box plots
	Figure B.5	MCC box plots
	Figure B.6	HV box plots
Number-of-defects	Figure B.7	Total # of S2 defects histograms
	Figure B.8	Total # of S3 defects histograms
	Figure B.9	Total # of S4 defects histograms
	Figure B.10	Total # of defects histograms
	Figure B.11	Total # of S2 defects box plots
	Figure B.12	Total # of S3 defects box plots
	Figure B.13	Total # of S4 defects box plots
	Figure B.14	Total # of defects box plots
Defect densities	Figure B.15	DD_2 histograms
	Figure B.16	DD_3 histograms
	Figure B.17	DD_4 histograms
	Figure B.18	DD histograms
	Figure B.19	DD_2 box plots
	Figure B.20	DD_3 box plots
	Figure B.21	DD_4 box plots
	Figure B.22	DD box plots
Amount-of-reuse	Figure B.23	ERD_2 histograms
	Figure B.24	ERF_2 histograms
	Figure B.25	ERL_2 histograms
	Figure B.26	ERD_2 box plots
	Figure B.27	ERF_2 box plots
	Figure B.28	ERL_2 box plots

Table B.1: Summery of figures in reference B

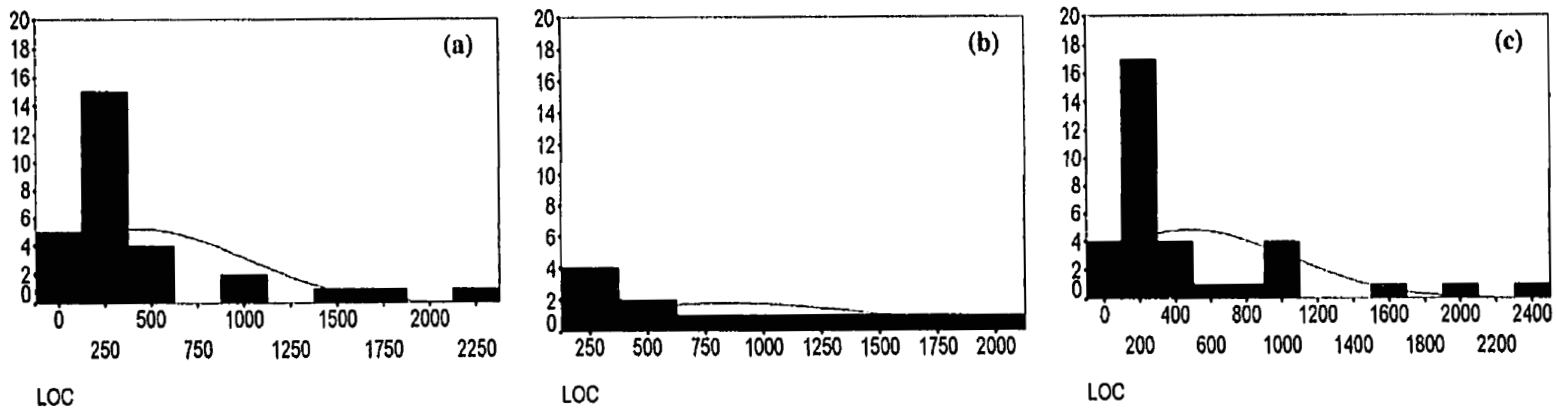

Figure B.1: LOC histograms for project 1 (a), project 2 (b), and project 3 (c)

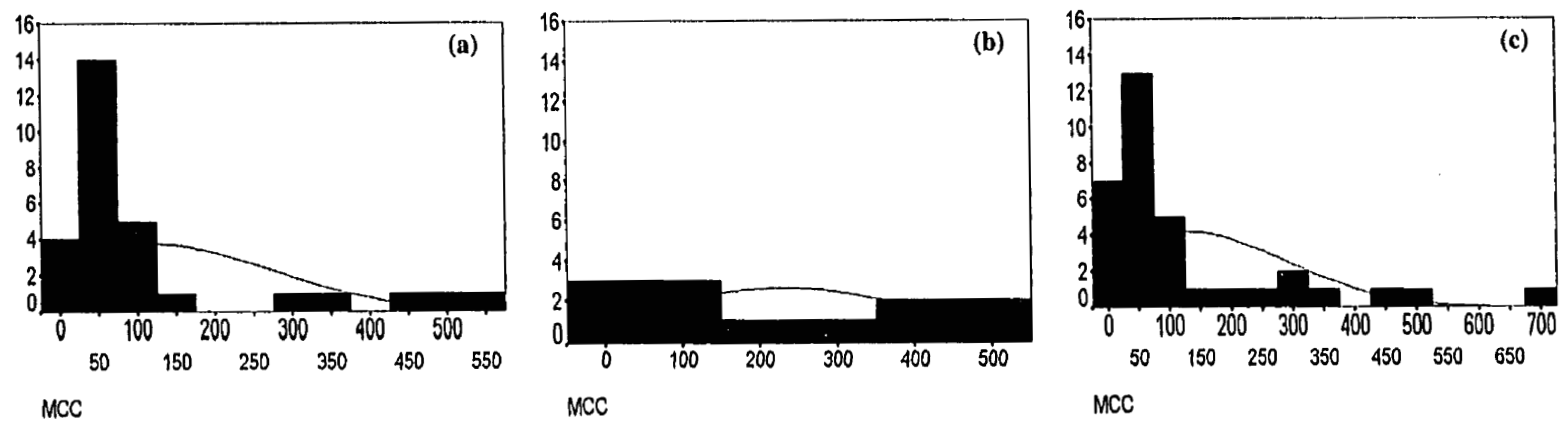

Figure B.2: MCC histograms for project 1 (a), project 2 (b), and project 3 (c)

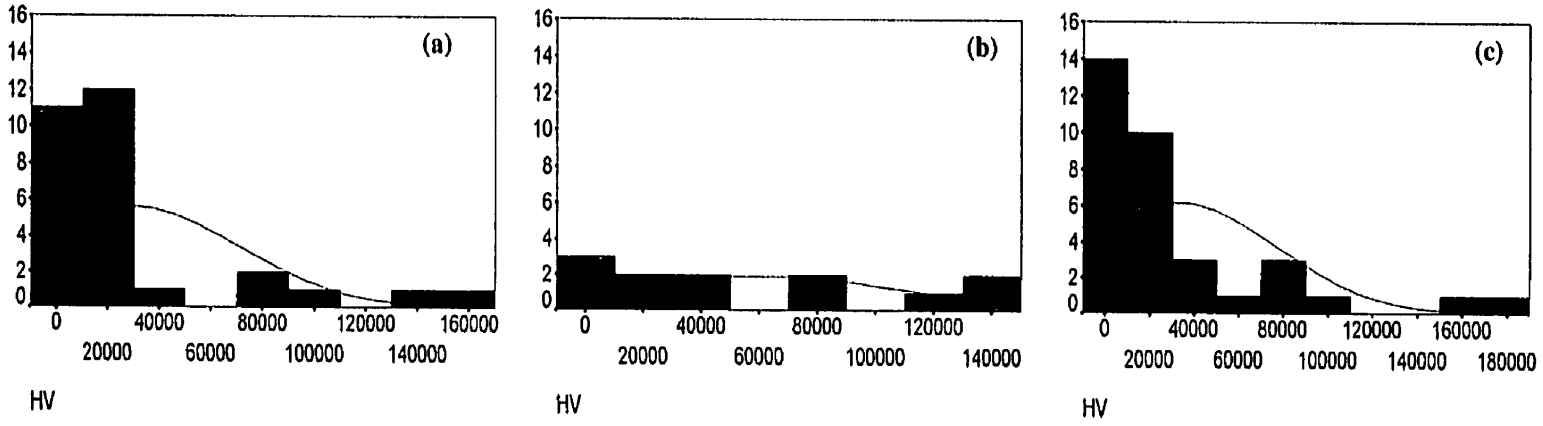

Figure B.3: HV histograms for project 1 (a), project 2 (b), and project 3 (c)

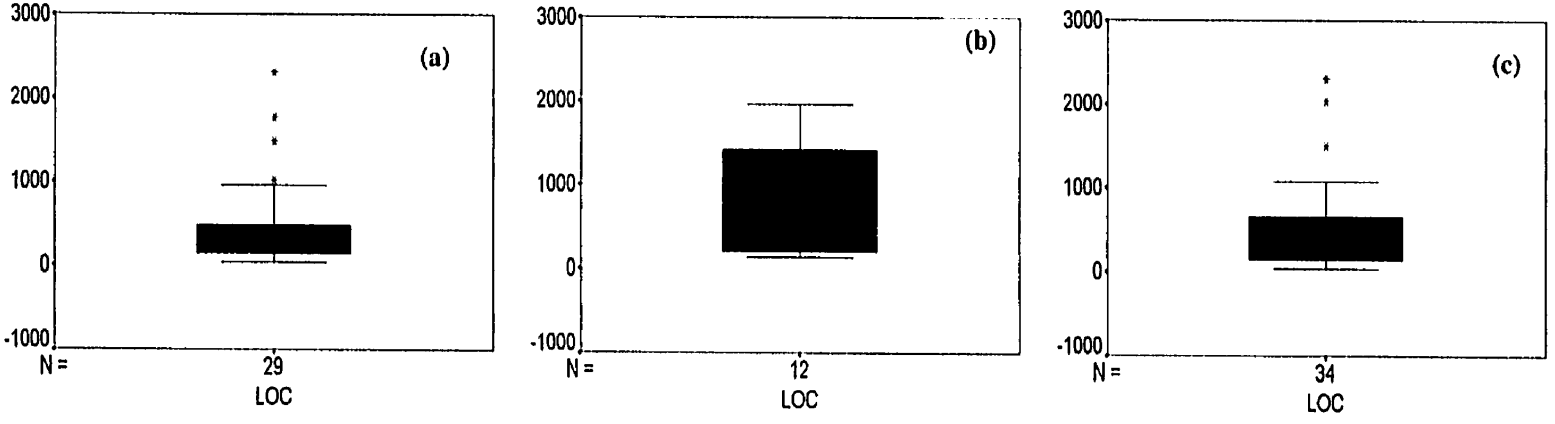

Figure B.4: LOC box plots for project 1 (a), project 2 (b), and project 3 (c)

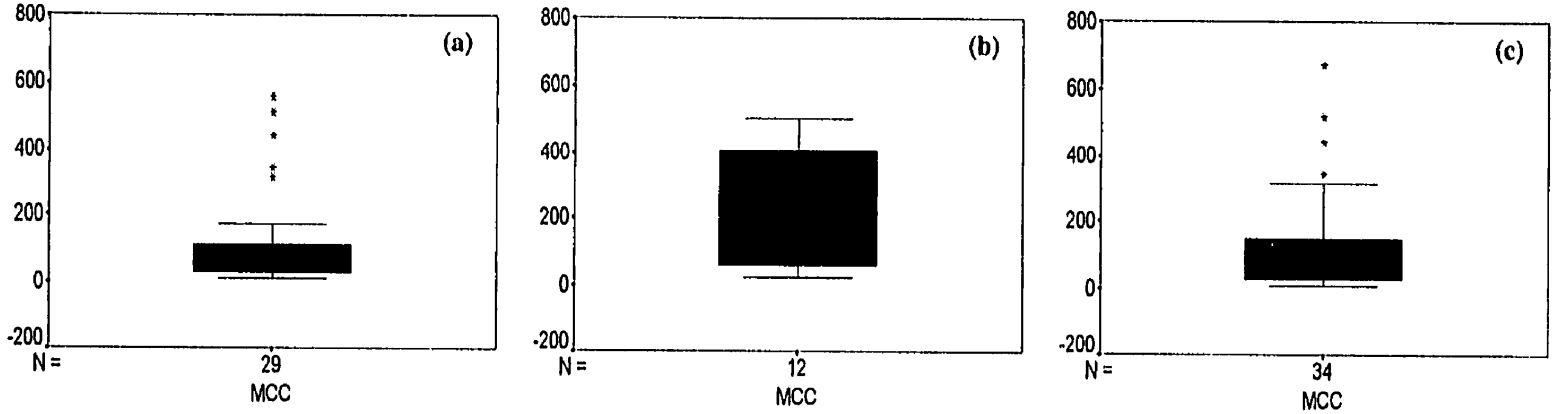

Figure B.5: MCC box plots for project 1 (a), project 2 (b), and project 3 (c)

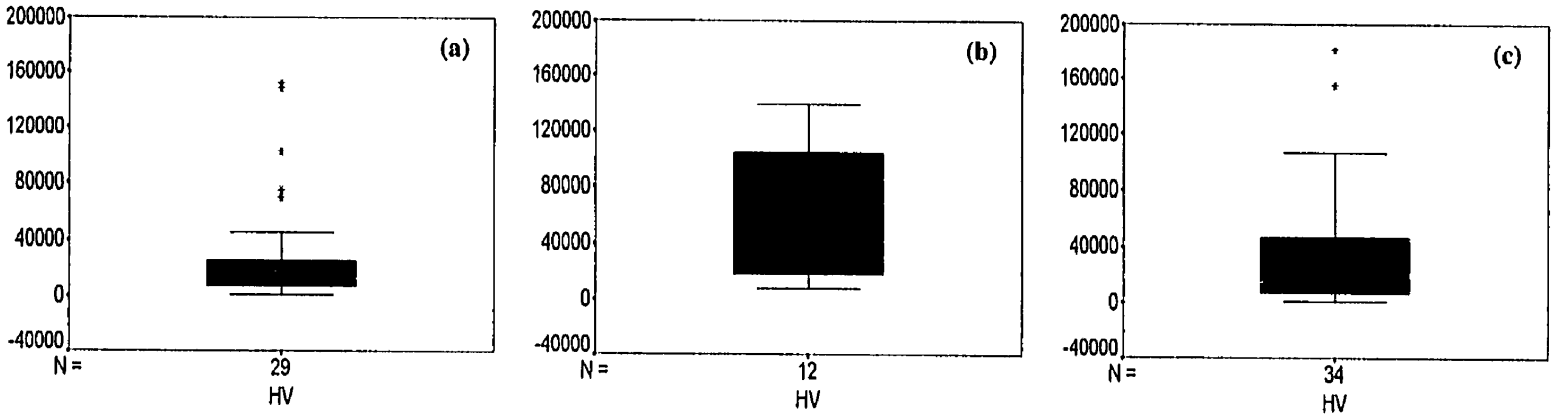

Figure B.6: HV box plots for project 1 (a), project 2 (b), and project 3 (c)

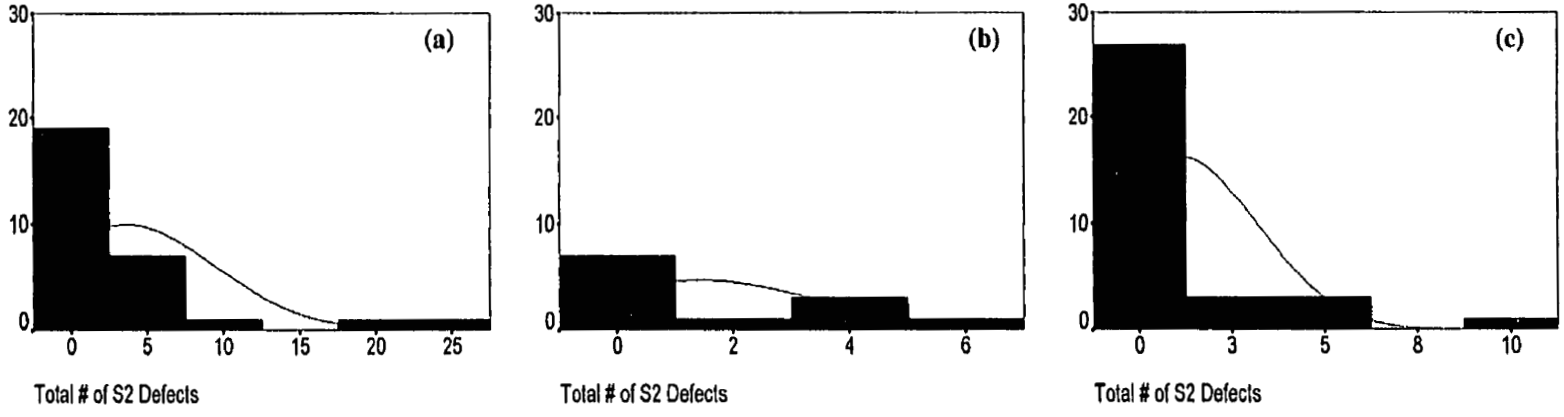

Figure B.7: Total # of S2 defects histograms for project 1 (a), project 2 (b), and project 3 (c)

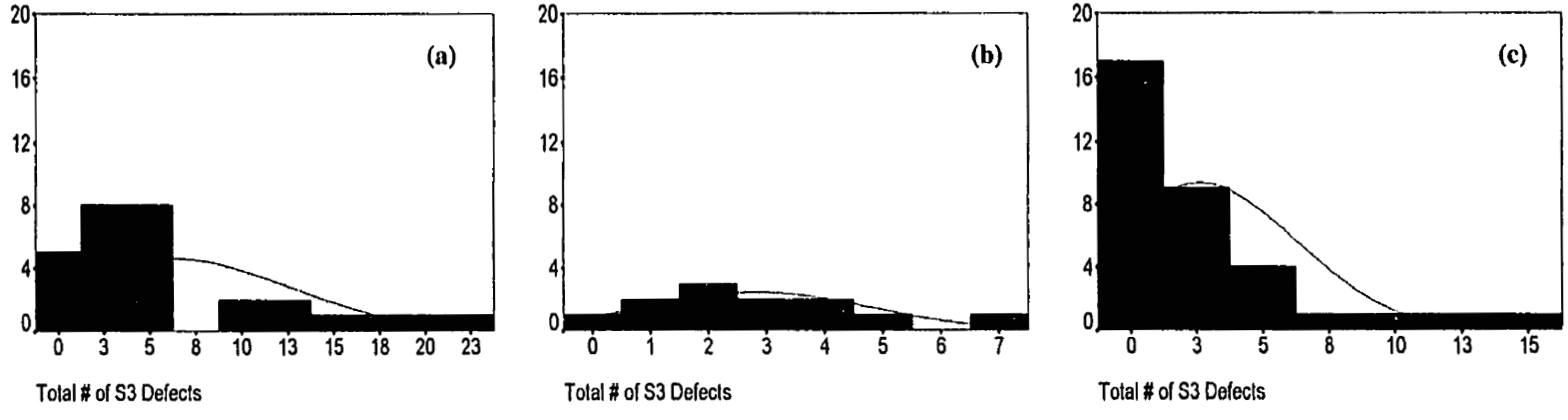

Figure B.8: Total # of S3 defects histograms for project 1 (a), project 2 (b), and project 3 (c)

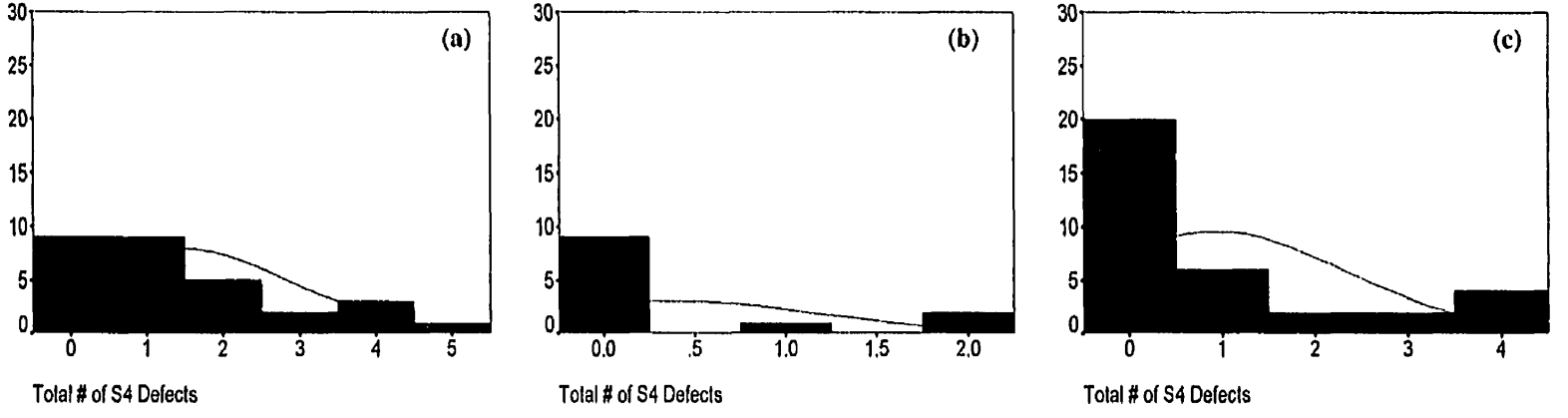

Figure B.9: Total # of S4 defects histograms for project 1 (a), project 2 (b), and project 3 (c)

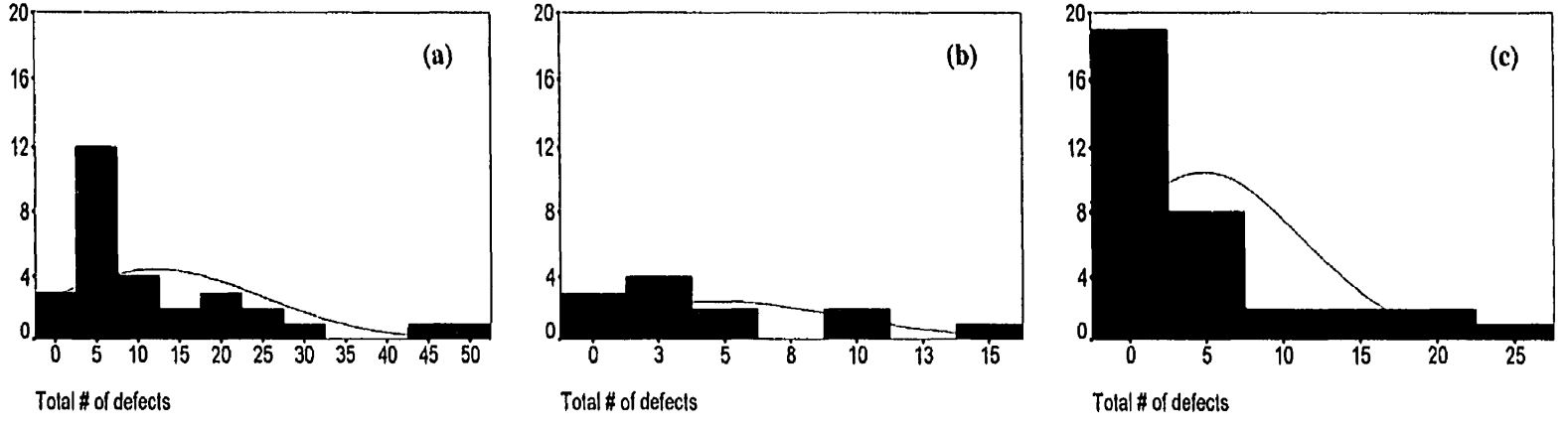

Figure B.10: Total # of defects histograms for project 1 (a), project 2 (b), and project 3 (c)

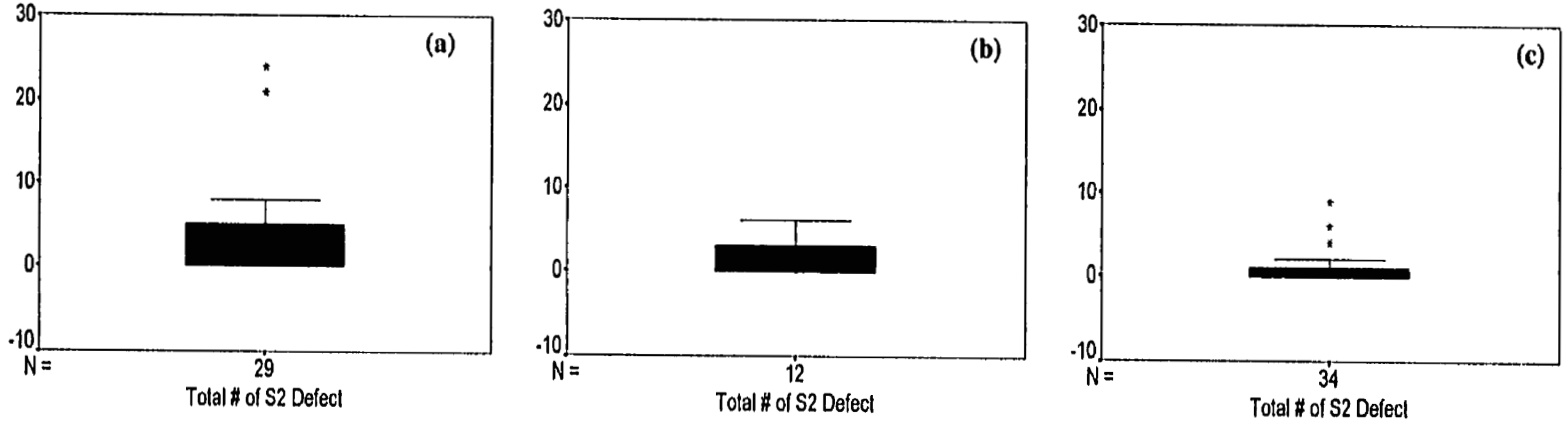

Figure B.11: Total # of S2 defects box plots for project 1 (a), project 2 (b), and project 3 (c)

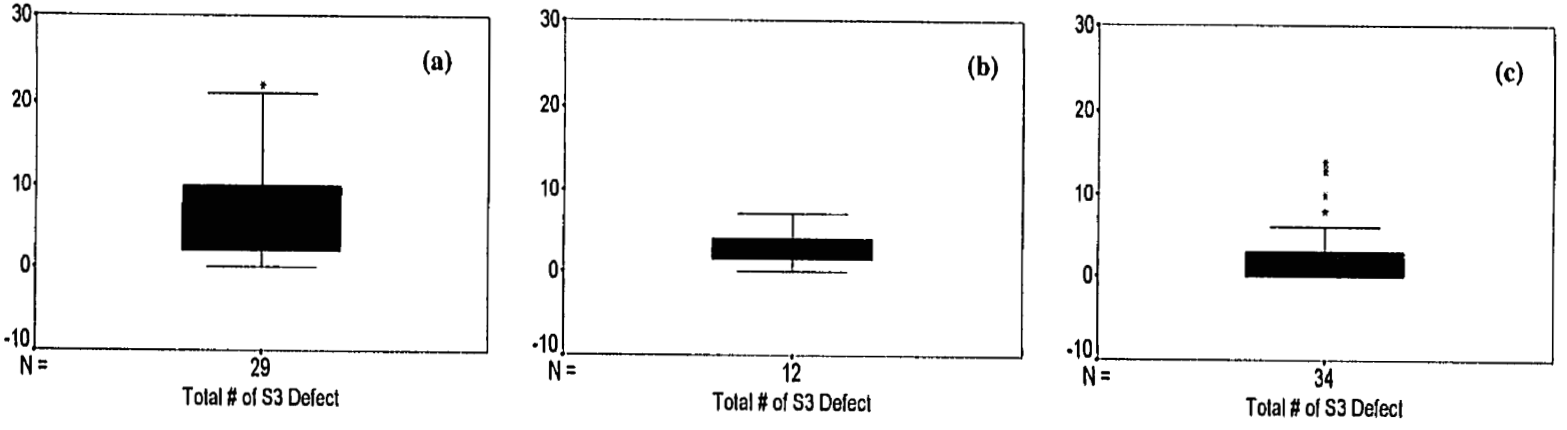

Figure B.12: Total # of S3 defects box plots for project 1 (a), project 2 (b), and project 3 (c)

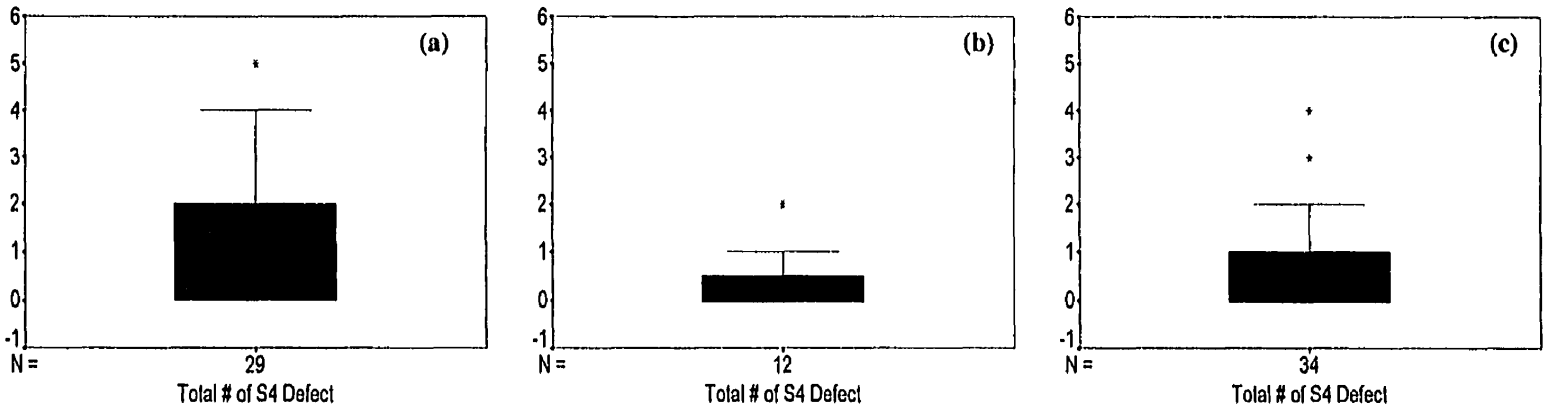

Figure B.13: Total # of S4 defects box plots for project 1 (a), project 2 (b), and project 3 (c)

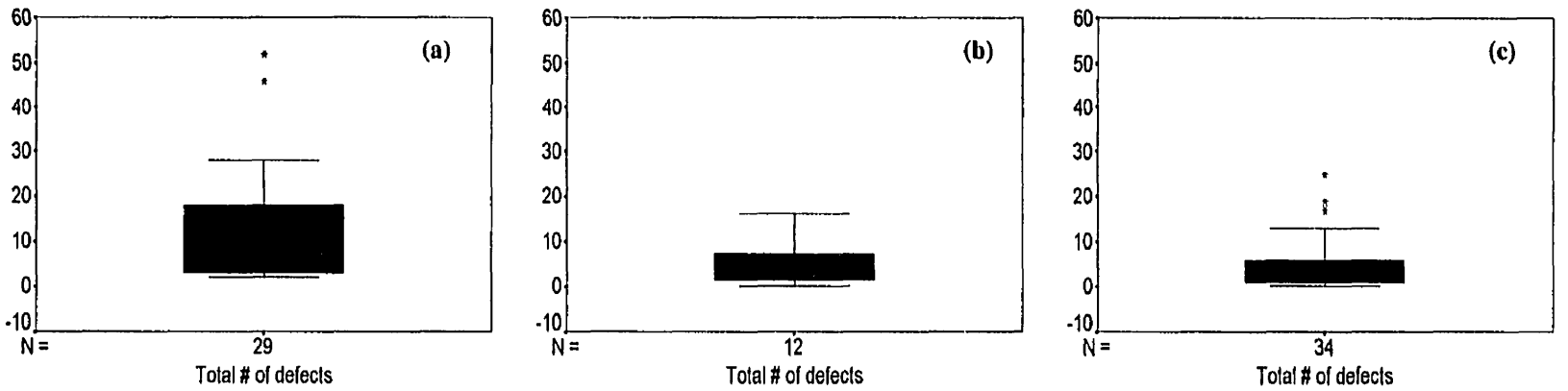

Figure B.14: Total # of defects box plots for project 1 (a), project 2 (b), and project 3 (c)

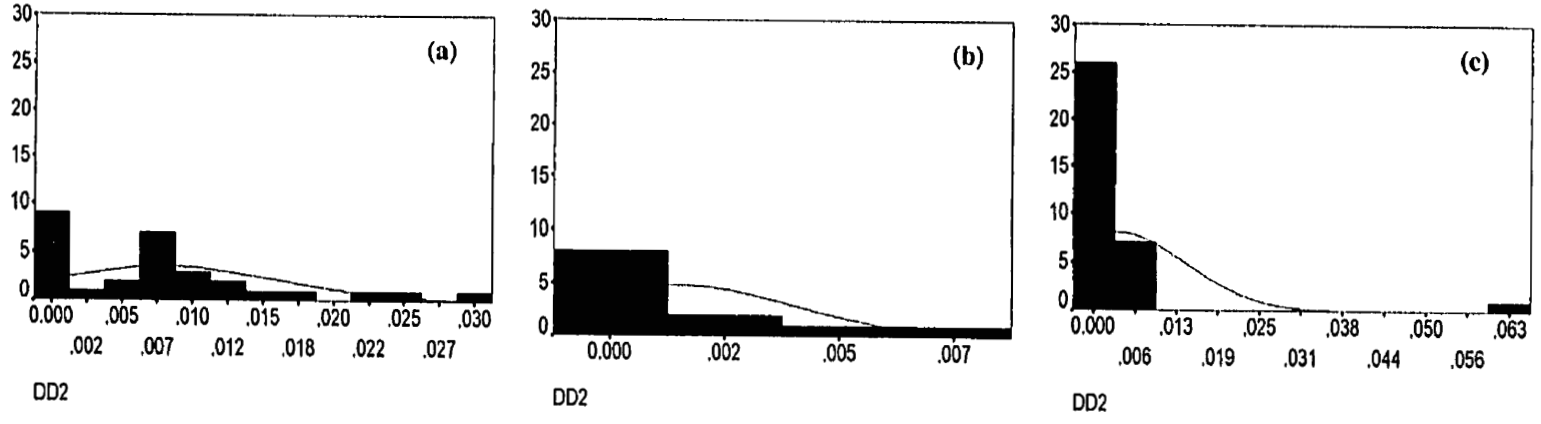

Figure B.15: DD_2 histograms for project 1 (a), project 2 (b), and project 3 (c)

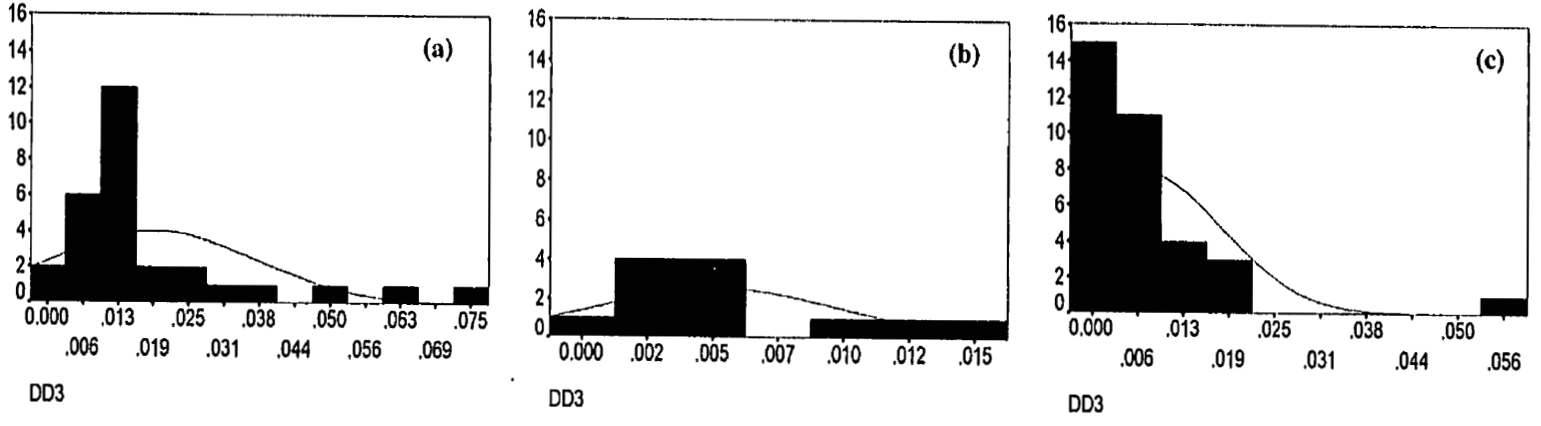

Figure B.16: DD_3 histograms for project 1 (a), project 2 (b), and project 3 (c)

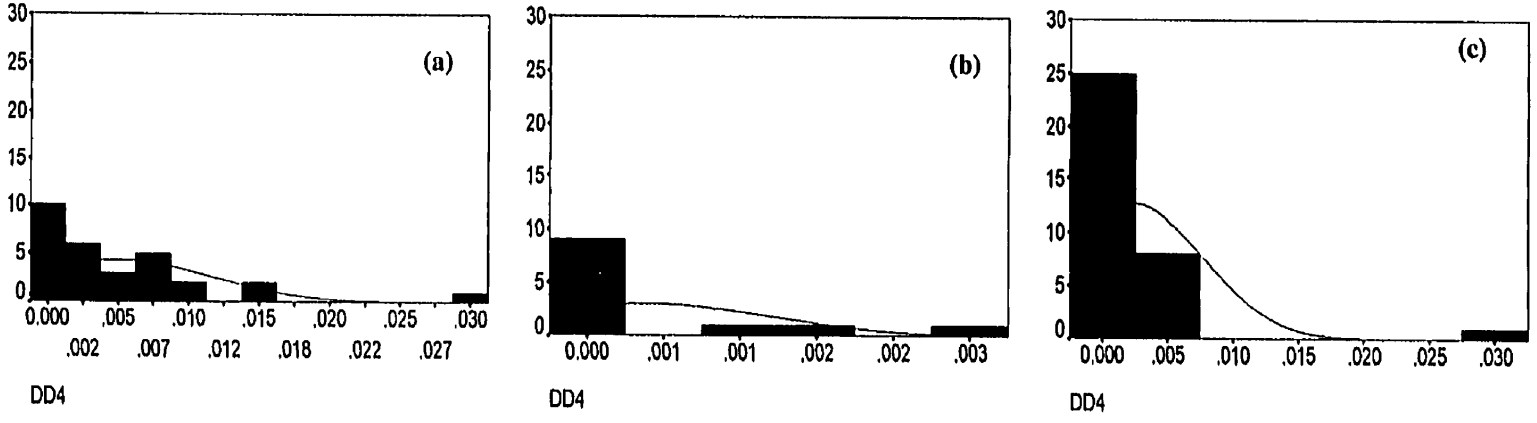

Figure B.17: DD_4 histograms for project 1 (a), project 2 (b), and project 3 (c)

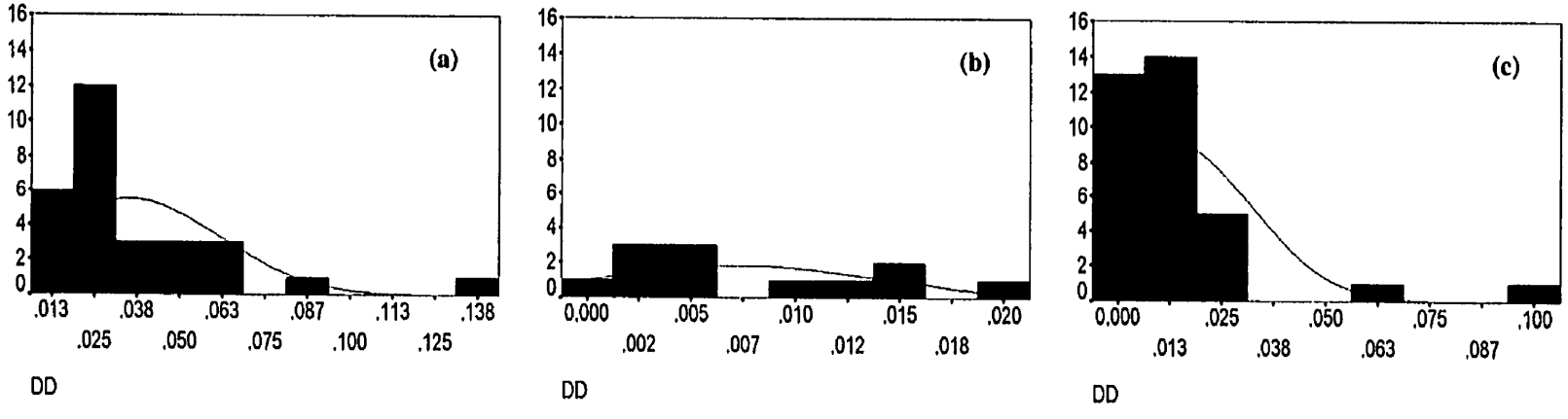

Figure B.18: DD histograms for project 1 (a), project 2 (b), and project 3 (c)

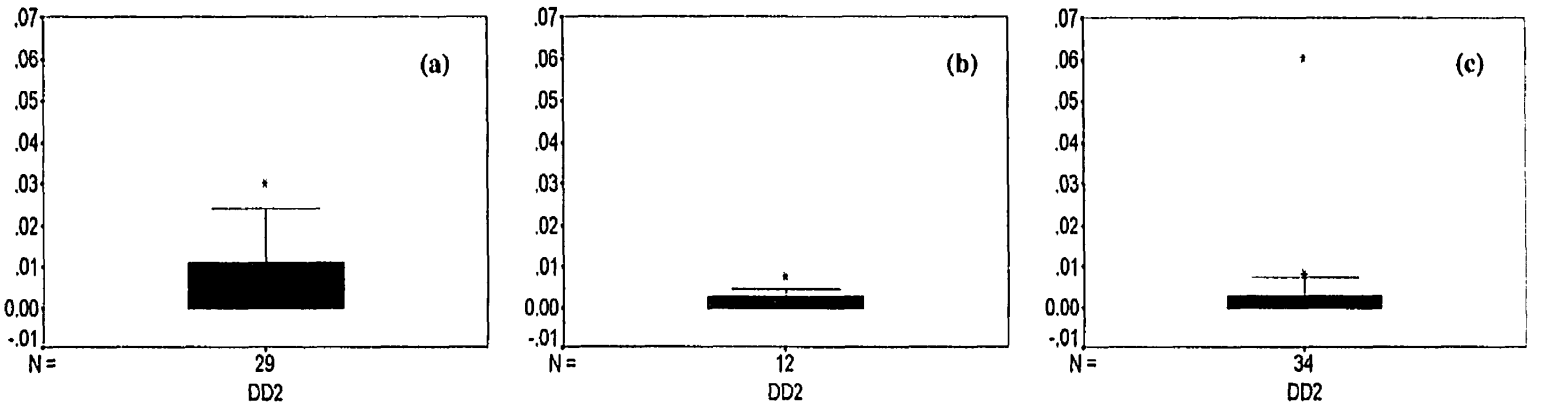

Figure B.19: DD_2 box plots for project 1 (a), project 2 (b), and project 3 (c)

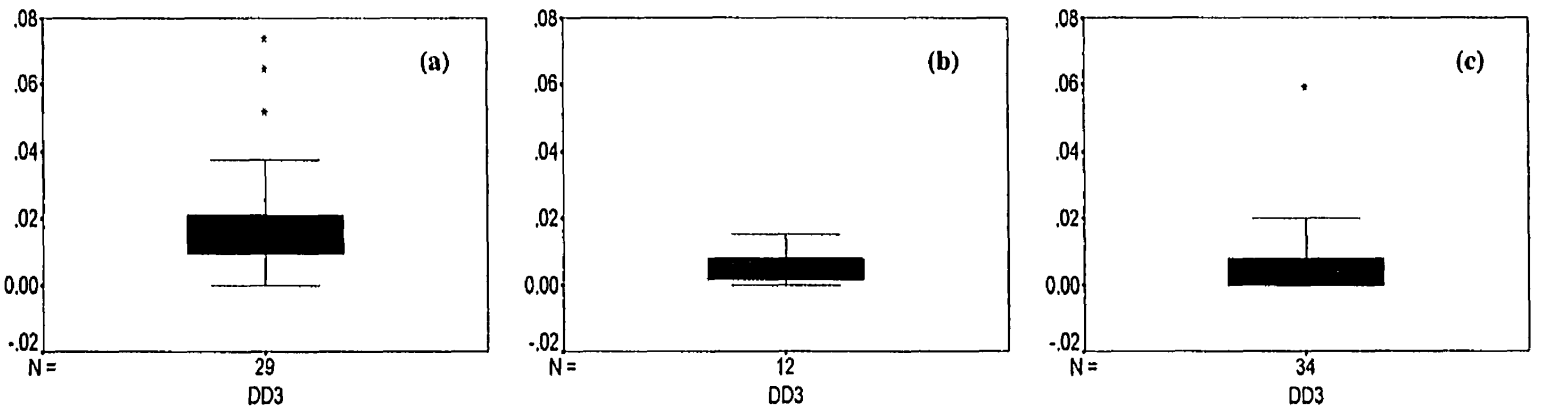

Figure B.20: DD_3 box plots for project 1 (a), project 2 (b), and project 3 (c)

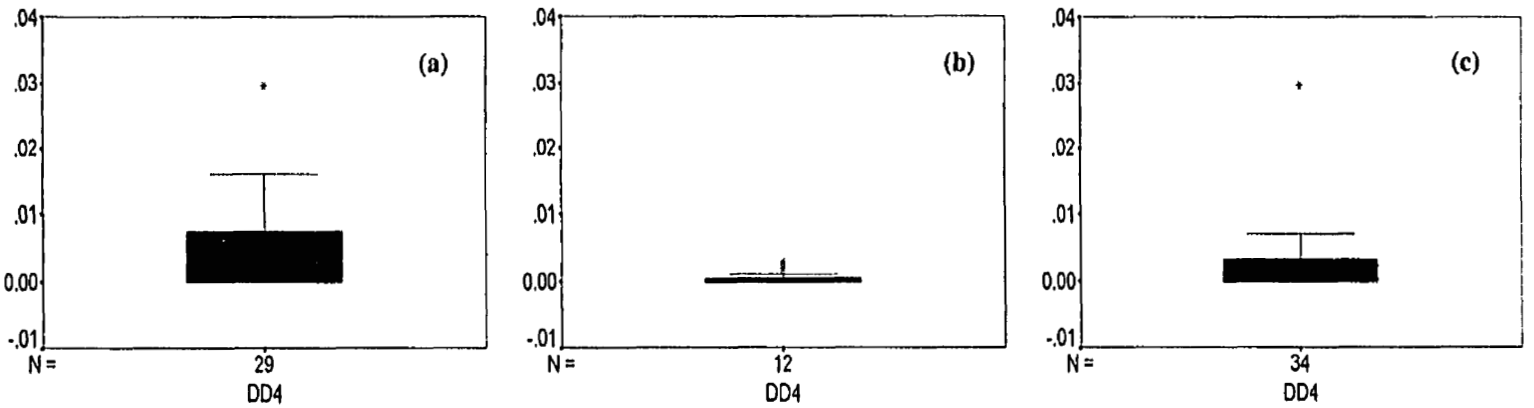

Figure B.21: DD_4 box plots for project 1 (a), project 2 (b), and project 3 (c)

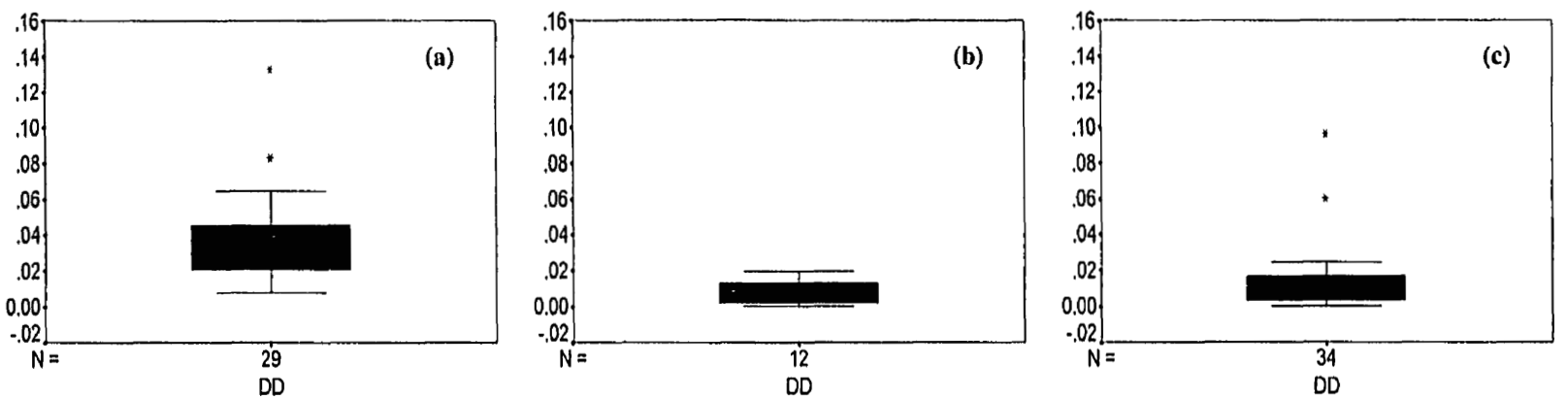

Figure B.22: DD box plots for project 1 (a), project 2 (b), and project 3 (c)

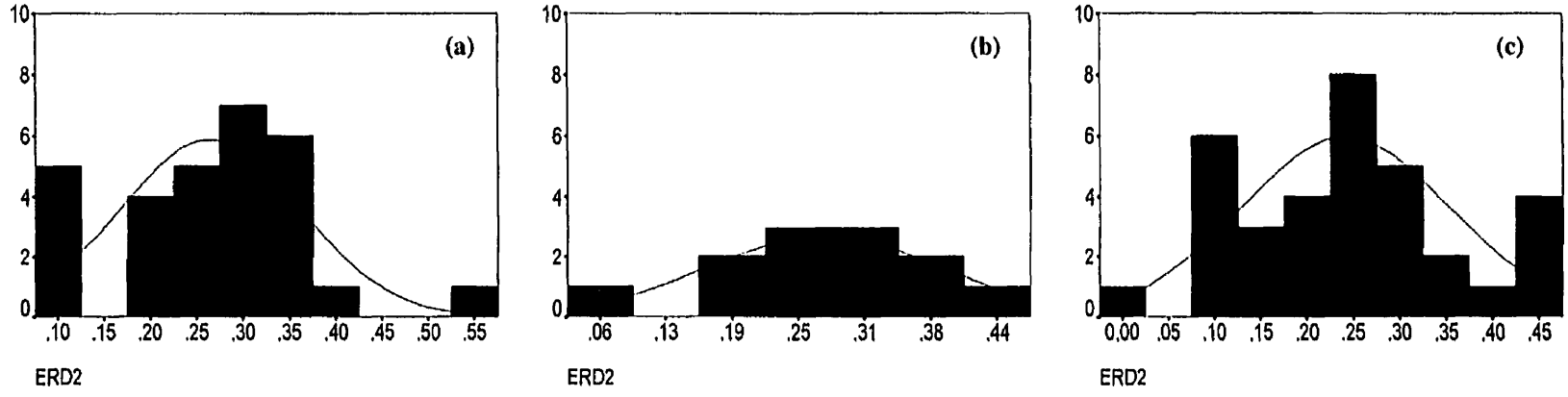

Figure B.23: ERD_2 histograms for project 1 (a), project 2 (b), and project 3 (c)

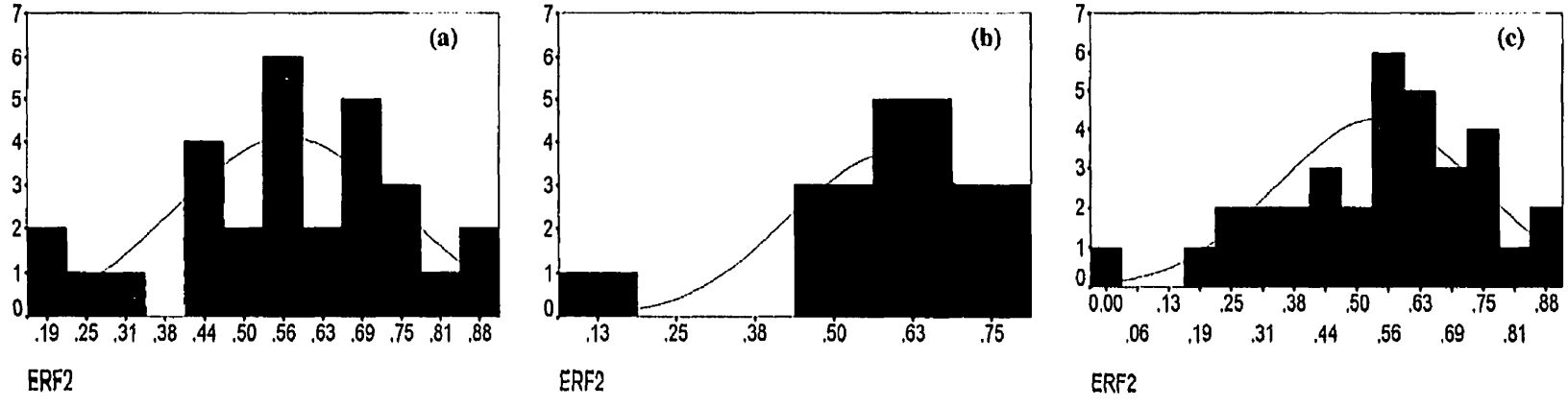

Figure B.24: ERF_2 histograms for project 1 (a), project 2 (b), and project 3 (c)

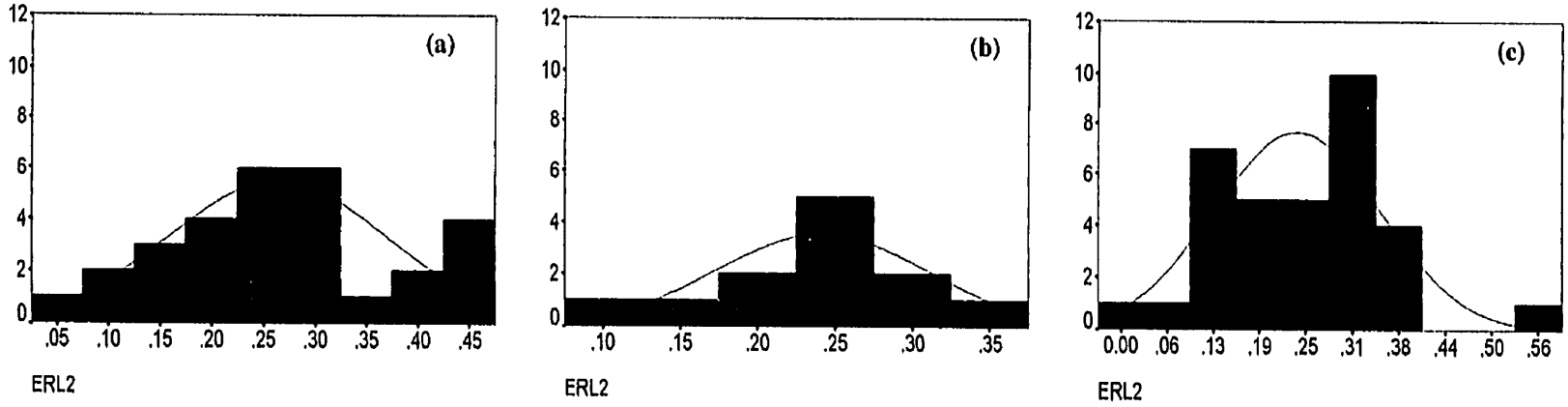

Figure B.25: ERL_2 histograms for project 1 (a), project 2 (b), and project 3 (c)

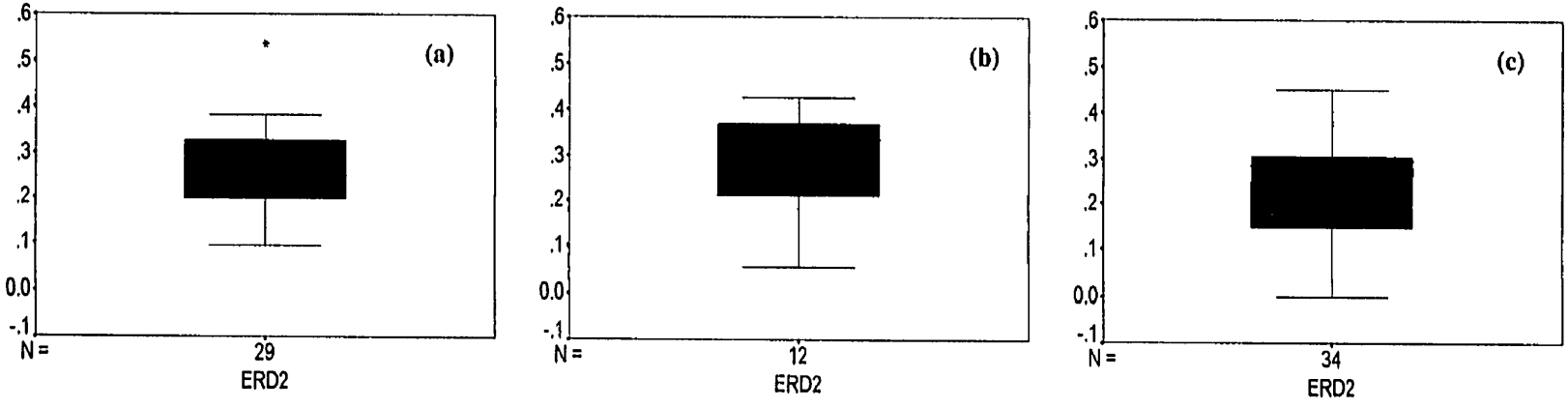

Figure B.26: ERD_2 box plots for project 1 (a), project 2 (b), and project 3 (c)

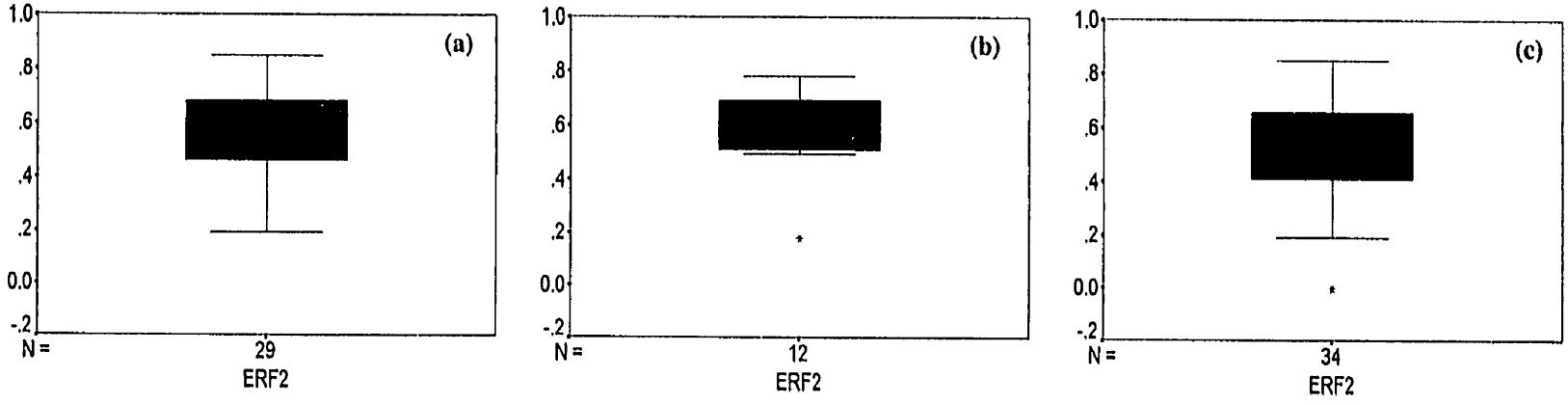

Figure B.27: ERF_2 box plots for project 1 (a), project 2 (b), and project 3 (c)

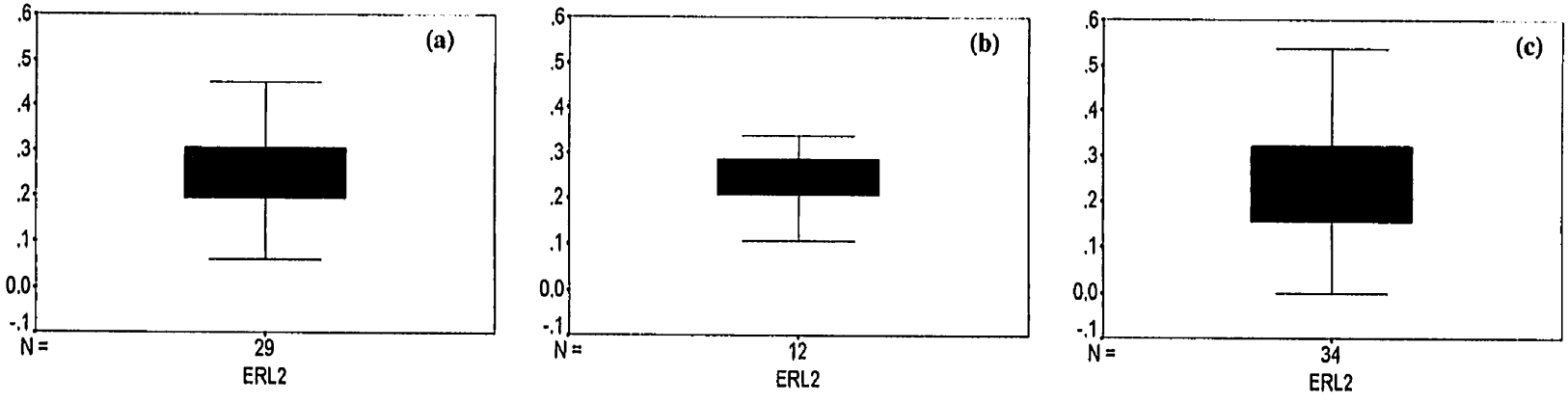

Figure B.28: ERL_2 box plots for project 1 (a), project 2 (b), and project 3 (c)