Predicting Software Maintainability by Using Object-Oriented Metrics

by

Melis Dagpinar
B.Eng., University of Istanbul, 1999

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming
to the required standard

[redacted]

Dr. J.H. Jahnke, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Storey, Departmental Member (Department of Computer Science)

[redacted]

Dr. H.A. Müller, Departmental Member (Department of Computer Science)

[redacted]

Dr. I. Traore,External Examiner(Department of Electrical and Computer Engineering)

Supervisor: Dr. Jens H. Jahnke

ABSTRACT

This thesis presents an empirical study designed for measuring the predictive power of object-oriented metrics for determining maintainability of object-oriented software systems. The study applies different kinds of measures applied to two selected systems. These measures can be grouped into four categories: size, inheritance, cohesion, and coupling. Unlike most related studies, indirect coupling has also been taken into account in order to analyze its usefulness. Maintainability characteristics have been derived from the systems' repository logs by categorizing the logs using three groups: perfective/adaptive, corrective, and preventive. Through analyzing the relationships between metrics and maintainability characteristics by using different statistical techniques, we have found that size and import direct coupling measures are significant predictors for measuring maintainability of a class while inheritance, cohesion, and indirect/export coupling measures are not.

Examiners:

[redacted]

Dr. J.H. Jahnke, Supervisor (Department of Computer Science)

[redacted]

Dr. M. Storey, Departmental Member (Department of Computer Science)

[redacted]

Dr. H.A. Müller, Departmental Member (Department of Computer Science)

[redacted]

Dr. I. Traore, External Examiner (Department of Electrical and Computer Engineering)

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to express my gratitude to all those who helped me to complete this thesis. I would like to thank the Fujaba developers for giving me permission to use their software system. I would also like to thank the KLOCwork Solutions Incorporation for assisting me and allowing me to use their reverse engineering tool along with its repository.

I am deeply indebted to my supervisor Dr. Jens. H. Jahnke whose help, stimulating suggestions and encouragement helped me greatly during my time of study at the University of Victoria.

Especially, I would like to give my thanks to my husband Yucel whose patient love enabled me to complete this work, and to my son Can Toprak, whose love made me so strong and so happy.

This thesis is dedicated to my wonderful parents Sevim and Zafer Dagpinar.

Chapter 1

Introduction

1.1 Motivation

Object-oriented design and development are popular concepts in today's software development environment. With the growing complexity and size of software systems, the ability to reason about quality characteristics is increasingly important. Maintainability is a particularly important quality characteristic as it is generally recognized that this is the most expensive stage of the software development process [1] [2]. Using metrics seems to be the only option to measure quality characteristics objectively and automatically.

The raising number of users of object-oriented languages has increased the demand for metrics that measure the quality of this type of software. Creating object-oriented metrics brings together many areas of study, from software development to statistics, and from project management to application domains. This variety causes metrics to be used for diverse purposes. For instance, while software developers use metrics for building better designs for their projects at the design level, project managers use metric results for the validation and quality of the software for marketing purposes after the software is completed. Moreover, at the design level, a company can use metrics in order to estimate the approximate number of days that will be spent on the project. This allows the company to then decide if more

employees are necessary for the completion of the project in the scheduled time. Consequently, both for making strategic decisions during each development phase and for marketing the software system, metrics are useful in determining the system's characteristics.

Maintainability is one of the most important quality characteristics. Unfortunately it is also probably one of the most difficult characteristics to measure. As some indicators (such as documentation) are likely to be inaccessible when we examine legacy systems. Most software programs need to be changed to meet the changing requirements of users and customers during the program's lifetime. It is generally impractical and uneconomical to produce software, which does not need to be changed. Introduction of new hardware also necessitates changes in the software.

Despite the fact that software maintenance is a very important and challenging task, it is poorly managed. One reason for poor management is the lack of a good measure of software maintainability [4]. Measuring maintainability is especially important for large legacy software systems, as they have evolved over time [3]. Evolution patterns include modifications related to the implementation, the interfaces and the overall system structure. Consequently, maintainability tends to degrade over time unless particular attention is paid to measure, assess and evaluate the effects of the evolution activities and effort spent on preventive maintenance tasks, refactoring and redesigning the code without changing its current functionality.

During our investigation of different metric studies, we have found that there is confusion on the terminology involved, especially for metric and measure. In our

explanations, we have used the term metric in order to denote quantitative results of represented formulas while we have used the term measure to denote formulas according to definitions in [15]. The reader should note, however, that while we were referencing papers, studies, and reports written by other authors, we have not made any change to the authors' use of metric and measure terms.

In this thesis, the maintainability of software systems has been investigated using only metrics derived from the selected systems without consulting their documentation. Unfortunately selecting suitable measures has been a difficult task due to current problems in the field of object-oriented metrics.

One problem in object-oriented metrics literature is ambiguity. As the goal of new metrics is not given clearly, there are not many empirical studies that support stakeholders in drawing firm conclusions about their usefulness. This impedes the adoption of metrics.

When we consider the object-oriented metrics literature, there are many new ideas and measurements. However, we do not see much improvement in the area of object-oriented measurements due to insufficient empirical studies to support the researchers' claims. Developers are reluctant to rely on unproven techniques and so they often attempt to create their own measurements.

1.2 Background

1.2.1 Object-oriented Measures

The fields of software engineering and computer science have been familiar with the term metrics since 1970. In the middle of the 70s, the two most famous metrics were presented. While McCabe's cyclomatic complexity technique looks at a program's control flow graph and the minimum number of paths in that graph [5], Halstead's "Number of distinct operators and operands" metric is used for complexity and measurement of programming effort [6]. Rocacher was one of the first to propose object-oriented metrics in 1988 since traditional metrics did not apply sufficiently to object-oriented programming languages [7]. The features for object-oriented languages can be given as following [8]:

- Encapsulation/Information Hiding
- Inheritance
- Polymorphism/Dynamic Binding
- All pre-defined types are objects
- All operations performed by sending messages to objects
- All user-defined types are objects

One of the most important aspects of object-oriented metrics is the ability to focus on the combination of function and data in terms of integrated objects. In contrast, traditional metrics, also called conventional metrics, measure the design structures and/or data structures independently.

Fenton has classified software metrics as the following [9]:

- Process metrics: Measure characteristics of the software process such as development, maintenance, testing. Typical process metrics are "effort involved", "costs incurred", or "elapsed time".
- Product metrics: Measure characteristics of the software product such as component, programs, and databases. Typical product metrics are complexity and various quality attributes.
- Resource metrics: Measure characteristics of the software resources such as hardware, software, or personal. Typical resource metrics are performance, availability, and productivity.

Fenton also distinguishes software metrics as internal (examining only the software) and external (examining the software in a environment).

1.2.2 Good Metric and Quality Characteristics

One important property that metrics should have is that their computation time should be as short as possible since time is a scarce resource in today's competitive business domain. This property should not be dismissed, especially for cohesion and coupling metrics that take indirect interactions into account, as they are the most expensive metrics to collect.

Elaine Weyuker defined a number of properties that good metrics should have [10]. However, Elaine Weyuker's definition of good metrics targets traditional

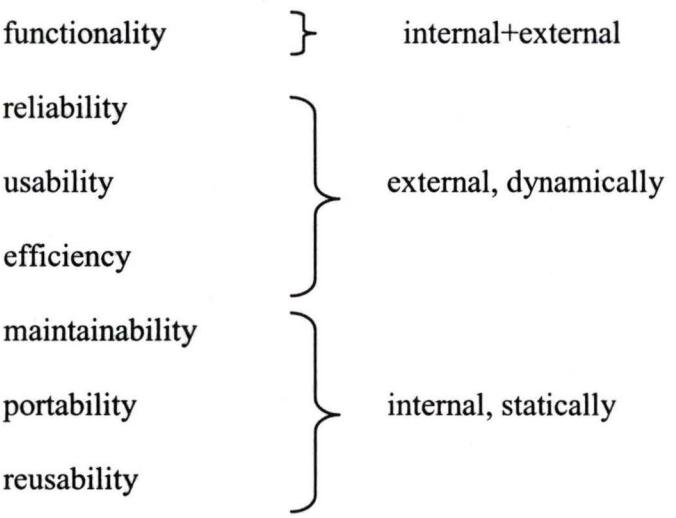
metrics only. Expert software engineers have discarded some of these traditional properties when object-oriented software is considered:

- Non-coarseness: The metric must be able to yield different results for different programs. A measure that says all programs are equally complex would be useless
- Non-uniqueness: The metric should not be too "discriminative either", i.e., the metric can yield the same result for different programs.
- Monotonicity: The metric value of the combination of two programs can never be less than the metric values for either of the two programs.
- Importance of Implementation: The metric should be able to distinguish among different implementations of the requirement. It means that metrics must depend to some extent on the details of the implementation.
- Non-equivalence of interaction: Metric value of programs (P+R) can be different from the metric value of programs (Q+R) even if metric value of P and Q are equal.
- Interaction increases complexity: Measure of a program formed by concatenating two programs should be greater than the sum of their measures

Briand *et al.* recommend a table to be prepared in order to avoid ambiguity for new metrics [11]. This table should include: a name, a definition, a classification whether or not it is operationally defined (yes or no), the level of objectivity (objective or subjective), a level of measurement (nominal, ordinal, interval), partially usable, usable, language specific, validation, and source parts.

There are standards for quality characteristics of software products. Therefore, object-oriented measures should be related to at least the one of those properties [12]:

ISO standard 9126 product characteristics



Other standards are [13]:

ANSI/IEEE standard 982 Dictionary of measures to produce reliable software (1998)

ANSI/IEEE standard 1061 for a software quality metrics methodology (1998)

ANSI/IEEE standard 1219 for software maintenance (1998)

1.2.3 Maintainability

When we look at the software maintenance literature, there are many definitions for maintainability. For instance, one of the most popular maintainability definitions is:

> *"The set of attributes that bear on the effort needed to make specified modifications."*

according to ISO 9126 [12]. The activities that cause those modifications can be categorized as following:

- **Analyzability:** Attributes of software that affect the effort needed for diagnosis of deficiencies or causes of failures, or for identification of parts to be modified.
- **Changeability:** Attributes of software that affect the effort needed for modification, fault removal or for environmental change.
- **Stability:** Attributes of software that affect the risk of unexpected effect of modifications.
- **Testability:** Attributes of software that affect the effort needed for validating the modified software.

Although the importance of maintainability is well known in the field of software design, there is still a lack of effort spent in order to increase maintainability of systems [14]. This causes many problems, for example it impedes system evolution (e.g. adding, deleting, or changing any software attribute.). Consequently, this increases the time and money spent for systems. Probably one of the main reasons of why most software developers do not properly consider maintainability is a lack of management. Project managers have to be aware of the consequences of building non-maintainable systems, and should be responsible for emphasizing maintainability throughout the development life cycle.

Due to some convincing empirical studies that demonstrate the correlation between a system's metrics and its maintainability, maintainability prediction analysis are usually used to indicate the system's maintainability [16]. The parameters used

for these prediction analyses usually include mean time to repair, mean maintenance person-hours per repair, and unit removal/installation times. As these parameters are subjective, they inevitably give inaccurate results.

1.3 Thesis Outline

The aim of this thesis is to design and conduct an empirical study that shows the predictive power of different selected metrics with respect to the maintainability of object-oriented systems.

Chapter 2 presents the measures used in addition to the maintainability characteristics, which were derived from the log entries of the subject systems' repository. Measures can be categorized into four groups: size, inheritance, cohesion, and coupling. The main difference of this study from many other object-oriented metric studies is determining maintainability characteristics by using metrics.

Chapter 3 presents the implementation of the empirical study. This chapter describes how the empirical study was designed and how the validation of metrics was analyzed with respect to maintainability. Moreover it presents some hypotheses and questions that are discussed in Chapter 4.

Chapter 4 presents the empirical study results step by step and discusses the hypotheses and questions posed in Chapter 3.

Finally, Chapter 5 draws some conclusions regarding the experience acquired from the empirical study and presents some recommendations for future studies.

Chapter 2

Measures and Maintainability Characteristics

This chapter introduces the maintainability characteristics and measures that have been used in our empirical study.

2. 1 Maintainability

IEEE Standard Glossary defines maintainability as [15]

> "*the ease with which a software system or component can be modified to correct faults, improve performance or other attributes, or adapt to a changed environment*"

Although external attributes have already been defined for software design and there are many software measures, we are still not able to demonstrate the quality of software efficiently.

There is little scientific work to show the relationships between internal attributes and external attributes, such as software maintenance. The main reason for this situation is the difficulty of measuring maintenance effort. Although there are many empirical studies trying to show correlations between metrics and fault-proneness, we do not yet have a clear idea about how and if fault proneness indicates external attributes (i.e. maintainability). In order to estimate the maintenance costs of a software system, we also need to consider other characteristics in addition to fault

proneness. Consequently, in our study we have analyzed metrics that affect four maintainability characteristics.

There are four distinct types of software maintenance:

A. Corrective

1. Processing Failure: Bugs. The abnormal termination of a program forcing job cancellation, production of "garbage" in an outputted report or file.
2. Performance Failure: The average inquiry response time exceeds some limit set, or an error rate in the transaction processing is greater than that specified as permissible.
3. Implementation Failure: Inconsistencies or incompleteness in the detailed design.

B. Adaptive

1. Change in data environment: A change in the classification code system associated with a particular data element, or the logical restructuring of a database.
2. Change in processing environment: The installation of a new generation of system hardware.

C. Perfective

1. Processing inefficiency: The use of an inferior computational algorithm, or inappropriate language features, or making poor use of the computer operator's time.

2. Performance Enhancement: Improving the readability of a report through reformatting.
3. Maintainability: Making the program more readable through the insertion of certain comments or improving the documentation.

D. Preventive: Updating the software in order to improve upon its future maintainability without changing its current functionality.

The terms perfective, adaptive, and corrective were originally used by Swanson [16]. Preventive maintenance is done to improve the future maintainability of the software system [17]. This type of maintenance is also referred to as preservation of the software system [18], where preservation involves using development resources to improve an aging system that often has a poor design and is therefore hard to maintain.

According to the above definitions, an improvement of maintainability reduces the effort when a change is made to program but does not affect the frequency of failures [19]. However, we believe that maintainability also affects the frequency of failures as the subject system is object-oriented and changing one object often greatly affects other objects.

The cost of the maintenance processes is not distributed evenly across all categories of maintenance. Studies by Leintz and Swanson [19] show that 50% of the total maintenance costs can be attributed to perfective maintenance, 25% for adaptive maintenance, whereas only 21% for corrective maintenance and only 4% for

preventive maintenance. Although these are relatively old studies, more recent analyses concur with these findings [20].

2.2 Measures

The measures that we have used for our empirical study can be categorized into four groups. The following section explains these measures.

2.2.1 Measuring Size

Many empirical studies use size metrics in order to show their correlation with other internal metrics and fault-proneness [21]. Consequently, we have a lot of information about the meaning of size metrics. For example, it has been shown that the size of a software system positively correlates with its fault-proneness. Therefore in addition to using size metrics for finding correlations with maintainability, size metrics can be used to show the validity of log files derived from the selected system repository. This is important because sometimes log files are not appropriately generated due to insufficient management of the software development procedure. In this case, the conclusion that log files are appropriate can be derived if a significant correlation can be found between size metrics and number of log entries.

One of the most used size measures is Lines Of Code (LOC), the number of lines containing source code. LOC measures the number of physical lines in the code. This is highly biased as it does not account for different coding styles, and it is also language dependent. The language dependency bias is not an important consideration

for our empirical study since the selected system releases are not written in different languages. Therefore, we have selected other size measures to avoid biased results due to different coding styles.

The first size measure is NIM [21], the number of instance methods in a class. This measure counts all the public, protected, and private methods defined for the instances of a class, and gives an idea of the size of the class interface.

The second size measure is TNOS, the total number of statements in a class. This measure is an extension of one of the measures of Mark Lorenz and Jeff Kidd [21]. The reason for this extension is that while Lorenz and Kidd defined a measure called the number of statements per method, there is no measure counting statements for a class. Thus, unlike LOC, TNOS counts statements in a class without being affected by different coding styles.

2.2.2 Measuring Inheritance

One of the most frequently used inheritance measures in empirical studies is DIT (Depth of Inheritance Tree), proposed in the CK metrics suite [22]. Because DIT is unable to distinguish between classes having the same distance from the root class, it is usually used with NOC (Number Of Children), also proposed in the CK metric suite. NOC for a class is defined by calculating the number of child classes one level below the selected class. We have used AID [23] instead of DIT, since there is no description of how to deal with multiple inheritance situations in the DIT definition. AID, average inheritance depth of a class, was proposed by Henderson-Sellers to

resolve this ambiguity. The AID of a class is calculated by taking the average of distances between the class and its root classes. In our study, the AID of a class without any ancestors is 1 and it is increased by one for each lower level in the inheritance hierarchy. Therefore, both AID and NOC were used for this empirical study.

2.2.3 Measuring Cohesion

Most cohesion measures are based on LCOM (Lack of COhesion in Methods) proposed by Chidamber and Kemerer [22]. The original definition of LCOM was

"*The number of pairs of methods in the class using no attribute in common.*"

Because that definition is highly dependent on the number of methods in the class, Chidamber and Kemerer have changed the definition as follows:

> "*The number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative, however LCOM is set to zero.*"

Unfortunately this definition did not improve the precision for measuring cohesion. For example, as shown in Figure 2.1 LCOM is not able to distinguish between these two dissimilar entities [24].

At Figure 2.1: (a) A highly non-cohesive class LCOM(a) = 8 (b) A highly cohesive class LCOM(b) = 8

So, the definition of LCOM was evolved by Rosenberg as follows [25]:

LCOM* Definition: Calculate for each data field in a class what percentages of the methods use that data field. Average the percentages then subtract from 100%. Lower

percentages mean greater cohesion of data and methods in the class.

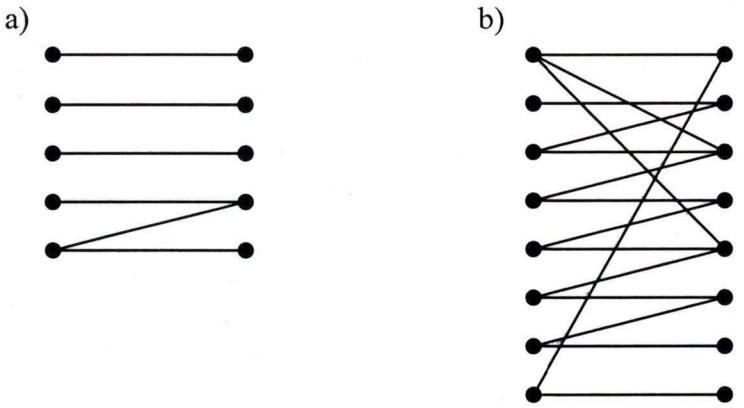


Figure 2.1. Method-variable interactions in a class

This definition allows better interpretation than the previous definitions from Chidamber and Kemerer since the result metrics range between a fixed interval of 0 to 1. Therefore, we can easily understand the cohesiveness of a class from LCOM*, while we cannot predict the cohesiveness of a class from LCOM. Consequently, LCOM*, is used in many empirical studies. However, LCOM* is still insufficient to distinguish between the classes shown in Figure 2.1 (LCOM* (a): 76%, LCOM* (b): 75%).

In order to be able to distinguish such classes, we also need to take indirect connections into account. The measure LCC (Loose Class Cohesion), defined by Bieman and Kang [26], deals with indirect connections. An instance variable is directly used by a method M, if the instance variable appears in the body of the method M. The instance variable is indirectly used if it is directly used by another method M' which is called directly or indirectly by M. Two methods are directly connected if they use, directly or indirectly, a common attribute. LCC counts the pairs

of methods that are, directly or indirectly, connected.

Let NP(C) be the total number of pairs of visible methods in a class C. NP is the maximum possible number of direct or indirect connections in a class. If there are N methods in a class C: NP(C) = N*(N-1)/2. Let NDC(C) be the number of direct connections and NIC(C) be the number of indirect connections in class C. Therefore LCC = (NDC(C)+NIC(C))/NP(C). For example in the class shown in Figure 2.2, the LCC = 1. This indicates a highly cohesive class while LCOM definitions are not sufficient in showing. The range of LCC is always in the [0,1] interval. Bieman and Kang propose three ways to calculate LCC: (1) include inherited methods and inherited distance variables in the analysis; (2) exclude inherited methods and inherited instance variables from the analysis; and (3) exclude inherited methods but include inherited instance variables.

```
class Stack {int * array, top, volume;
        public:
                Stack (int v) {
                        volume = v:
                        array = new int[volume];
                        top = 0;}
                int Isempty() {
                        return top==0};
                int Volume() {
                        return volume};
                int Vtop() {
                        return array[top-1];}
                void Push (int item) {
                        if (top==volume)
                                printf("Empty stack.\n");
                        else
                                array[top++]=item;}
                int Pop() {
                        if(Isempty())
                                printf("Full stack.\n");
                        else
                                --top;}  };
```

Figure 2.2. Sample class

In our study, we chose the second way to calculate LCC, since we believe that cohesion should show how well the class is built by itself, without the help of any other classes.

2.2.4 Measuring Coupling

The issues involved in measuring the coupling of a class are:

1. Inheritance vs. non-inheritance coupling
2. Import and export coupling
3. Direct and indirect coupling
4. Coupling relationship

The first argument about inheritance vs. non-inheritance coupling has been started by Chidamber and Kemerer's CBO (Coupling Between Object classes) definition [22]. While CBO was defined without inheritance relationships in the first CK metric suite, the definition has been restated to include inheritance relationship in the CK metric suite. Since then, many suggestions for measuring coupling have been made. For instance, the metrics presented by Rajaraman and Lyu [27] distinguish measuring coupling for inheritance and non-inheritance relationships. They have proposed four measures for coupling: CNIC (Class Non-Inheritance-related Coupling), CIC (Class Inheritance-related Coupling), CC (Class Coupling), AMC (Average Method Coupling). Even though they introduced CNIC and CIC for measuring inheritance-related coupling aspects separately, they have only used the sum of CNIC and CIC (called CC) for their empirical studies. Likewise, we haven't

seen many convincing empirical studies that emphasize non-inheritance coupling and inheritance coupling metrics should be calculated separately until the unified framework presented by Briand *et al.* [11].

When we review the coupling literature, we see that there are few differentiations between export coupling and import coupling. While we can define export coupling of class C as interactions of being used by other classes, the import coupling of class C is defined as interactions using other classes. While most measures are just for import coupling, we have also used the export coupling measure since we believe that there is a distinction between the server class (providing information) and client class (requesting information) from a maintenance perspective. For instance, a child class (client type class) is exposed to changes when its root class (server type class) is changed. In contrast, this affection is not likely to occur for a root class when its child class is changed.

At this point, empirical studies about direct and indirect coupling are given in [28] and [29]. If class A uses class B, and class B uses class C, the coupling between A and B is defined as direct coupling while the coupling between A and C is defined as indirect coupling. The authors of the studies found that direct coupling is a useful predictor, while indirect coupling is not. In our empirical study, we have also taken into account indirect coupling as opposed to their study since their framework only focuses on using coupling measures to identify fault-prone classes and they have not given a clear explanation about why indirect coupling metrics are not useful.

In this study, there are eight indirect coupling measures. Four of them are calculated using the total number of interacted classes, and the others are calculated using the total number of interactions taking into account indirect relationships. The following figure explains this with using an import-coupling example.

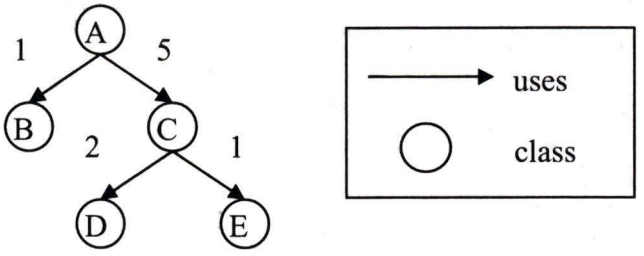


Figure 2.3. Basic example showing non-inheritance interactions between classes

As seen in Figure 2.3, there are indirect relationships between classes A and D, and A and E since class A is using C and class C is using D and E. Therefore NIIC (Non-Inheritance Import Coupling), TNIIC (Total Non-Inheritance Import Coupling), DTNIIC (Direct Total Non-Inheritance Import Coupling), and IDTNIIC (InDirect Total Non-Inheritance Import Coupling) are calculated as follows:

NIIC for class A: 1 + 5 = 6

TNIIC for class A: 1 + 5 + 2 + 1 = 9

DTNIIC for class A: 2 (B and C)

IDTNIIC for class A: 4 (B, C, D, and E)

*Please see Table 2.1, 2.2, 2.3, and 2.4 for an explanation about coupling measures.

The clear distinction between interactions is made by the framework proposed by Briand *et al.* According to these researchers [11], there are three types of interactions:

- Class-Attribute: There is a class-attribute interaction between classes c and d if class c is the type of an attribute of class d.

- Class-Method: There is a class-method interaction between classes c and d if class c is the type of a parameter of method m_d, or class c is the return type of method m_d.
- Method-Method: There is a method-method interaction between classes c and d if m_d directly invokes m_c, or m_d receives via parameter a pointer to m_c thereby invoking m_c indirectly.

The measures used in our empirical study are mainly based on measures defined at Briand *et al.* The changes made to the Briand measures are as follows:

- A separation has been made to distinguish between indirect and direct coupling.
- Relationship names have been changed from antecedent class, friend class, and other class to inheritance and non-inheritance relationships since friend class is only valid for C++ and we do not want to make our model language specific.
- In addition to counting interactions between classes, the number of classes that interact with a given class is also counted.

Therefore, the coupling measures are summarized in the following tables. Note that we have chosen so many different coupling measures for our experiment since we believe that coupling measures are the most promising indicators for the maintainability of a class.

Name	Definition
ICAIC	Inheritance Class-Attribute Import Coupling.
NICAIC	Non-Inheritance Class-Attribute Import Coupling.
ICAEC	Inheritance Class-Attribute Export Coupling.
NICAEC	Non-Inheritance Class-Attribute Export Coupling
ICMIC	Inheritance Class-Method Import Coupling.
NICMIC	Non-Inheritance Class-Method Import Coupling.
ICMEC	Inheritance Class-Method Export Coupling.
NICMEC	Non-Inheritance Class-Method Export Coupling.
IMMIC	Inheritance Method-Method Import Coupling.
NIMMIC	Non-Inheritance Method-Method Import Coupling.
IMMEC	Inheritance Method-Method Export Coupling.
NIMMEC	Non-Inheritance Method-Method Export Coupling.
IIC	Inheritance Import Coupling. IIC for a class is calculated by adding ICAIC, ICMIC, and IMMIC metrics for a class.
NIIC	Non-Inheritance Import Coupling. NIIC for a class is calculated by adding NICAIC, NICMIC, and NIMMIC metrics for a class.
IEC	Inheritance Export Coupling. IEC for a class is calculated by adding ICAEC, ICMEC, and IMMEC metrics for a class.
NIEC	Non-Inheritance Export Coupling. NIEC for a class is calculated by adding NICAEC, NICMEC, and NIMMEC metrics for a class.

Table 2.1. Direct coupling measures counting the number of interactions

Name	Description
TIIC	Total Inheritance Import Coupling by including indirect coupling relationships.
TNIIC	Total Non-Inheritance Import Coupling by including indirect coupling relationships.
TIEC	Total Inheritance Export Coupling by including indirect coupling relationships.
TNIEC	Total Non-Inheritance Export Coupling by including indirect coupling relationships.

Table 2.2. Indirect coupling measures counting the number of interactions

Name	Description
DTIIC	Direct Total Inheritance Import Coupling. DTIIC of class A is calculated by counting number of inherited and non-inherited classes used by A.
DTNIIC	Direct Total Non-Inheritance Import Coupling. DTNIIC of class A is calculated by counting number of non-inherited classes used by A.
DTIEC	Direct Total Inheritance Export Coupling. DTIEC of class A is calculated by counting number of inherited and non-inherited classes using A.
DTNIEC	Direct Total Non-Inheritance Export Coupling. DTNIEC of class A is calculated by counting number of non-inherited classes using A.

Table 2.3. Direct coupling measures counting the number of classes

Name	Description
IDTIIC	InDirect Total Inheritance Import Coupling. IDTIIC of class A is calculated by counting number of inherited and non-inherited classes used by A by taking indirect relationships into account.
IDTNIIC	InDirect Total Non-Inheritance Import Coupling. IDTNIIC of class A is calculated by counting number of non-inherited classes used by A . by taking indirect relationships into account
IDTIEC	InDirect Total Inheritance Export Coupling. IDTIEC of class A is calculated by counting number of inherited and non-inherited classes using A by taking indirect relationships into account.
IDTNIEC	InDirect Total Non-Inheritance Export Coupling. IDTNIEC of class A is calculated by counting number of non-inherited classes using A by taking indirect relationships into account.

Table 2.4. Indirect coupling measures counting the number of classes

2.3 Summary

This chapter presented the background for maintainability characteristics and measurements used in our empirical study.

Measures are categorized into four groups: size, inheritance, cohesion, and coupling. The focus was on coupling measurements since we believe they are of primary importance in predicting the maintainability characteristics for object-

oriented systems. We have tried to consider as many coupling measures as possible in order to investigate maintainability characteristics.

Chapter 3

Case Study Design

This chapter introduces the systems used in our empirical study and explains statistical techniques used to analyze relationships between metrics derived from the systems as well as their maintainability characteristics.

3.1 Systems

For this empirical study, two Fujaba subsystems, developed at the University of Paderborn, were used [30]. Fujaba is a round trip engineering tool that combines UML, SDM, Java and Design Patterns. In selecting the subsystems to study, a primary concern was the evidence of false positive results. This criterion leads to the selection of the subsystems *dobs*, containing 27 classes, and *uml* containing 178 classes. Subsystems with different sizes are more likely to have different metric results and maintainability than similarly sized subsystems. For example, classes in larger systems have bigger coupling metric results than classes in smaller systems. Moreover, it is easier to predict the maintainability characteristics of class A by comparing it with the maintainability characteristics of class B, found in a similar size system with class A when both metrics for class A and class B are similar. Therefore,

selecting different sized systems is important to capture universally predictive models.

Both the *uml* and *dobs* systems have been in development since January 1998. Metrics have been calculated for the period between January 1998 and April 2002 using three-month intervals. Relationships between selected metrics and maintainability characteristics were analyzed between January 1999 and April 2001. This period enabled us to study more number of classes, since not many classes have been created and developed between January 1998 and January 1999.

In our analysis, we have ignored interfaces, classes used for test purposes, and classes that had existed for less than 2 years as of April 2002.

3.2 Data Gathering

3.2.1 Collection of the Data from Subject Systems

The University of Paderborn has provided two subsystems for the empirical study. The class files and log files were downloaded using the dsd repository. Distributed Software Development, dsd, is a configuration and versioning toolkit for use in mobile and highly distributed software development groups. It is based on Sun's Jini Technology and the Concurrent Versioning System [32].

Logs were grouped using only three maintainability characteristics, as the two maintainability characteristics perfective and adaptive could not be distinguished given the log files available to us. For example, when a change is made to a file, we were not able to classify it as a perfective or adaptive maintainability characteristics

since we did not know if either customer requirements or hardware/software performance issues caused the change. Perfective/adaptive and corrective maintenance characteristics were stored for each class between the class's current developed time and April 2002. Preventive maintenance characteristics were stored between the time class was created and the class's current developed time, as preventive maintenance affects the class's future behavior rather than the class's previous behavioral characteristics.

3.2.2 KLOCwork

3.2.2.1 KLOCwork InSight

KLOCwork InSight tool's repository [31] was used to get appropriate information for the calculation of metrics. KLOCwork InSight is a reverse engineering tool adopted by both software developers and designers. It enables to them to extract behavioral and structural information from software source code. Extracted data is then mined by KLOCwork InSight and stored in a database. KLOCwork InSight provides the following tools that help users to analyze and develop software systems written in C/C++, Java and other programming languages:

- Web Communicator: allows users to view help documentation and user comments added for specified entities.
- Architect: presents graphical models of a system's structural and process architecture.
- Flowchart: displays the code associated with the source file.

- X-Ref: searches for and lists software system entities and the files.
- Class Browser: displays the class inheritance hierarchy of a group of C++ software entities within a selected project.

3.2.2.2 Metric information in KLOCwork InSight (Version 5.0)

As of Version 5.0, KLOCwork InSight does not contain any tool to work with metrics. Metrics are used in two new products, KLOCwork Software Analysis, and KLOCwork GateKeeper. However, those products do not have measures for object-oriented systems. Consequently all measures used, except TNOS and NIM, were calculated completely from scratch. When a project is first set up, the information about metrics is not included although software analysis reports are prepared automatically. Consequently, to use the metrics information provided by KLOCwork Software Analysis and KLOCwork GateKeeper, some manual adjustments and reloading of the database were required.

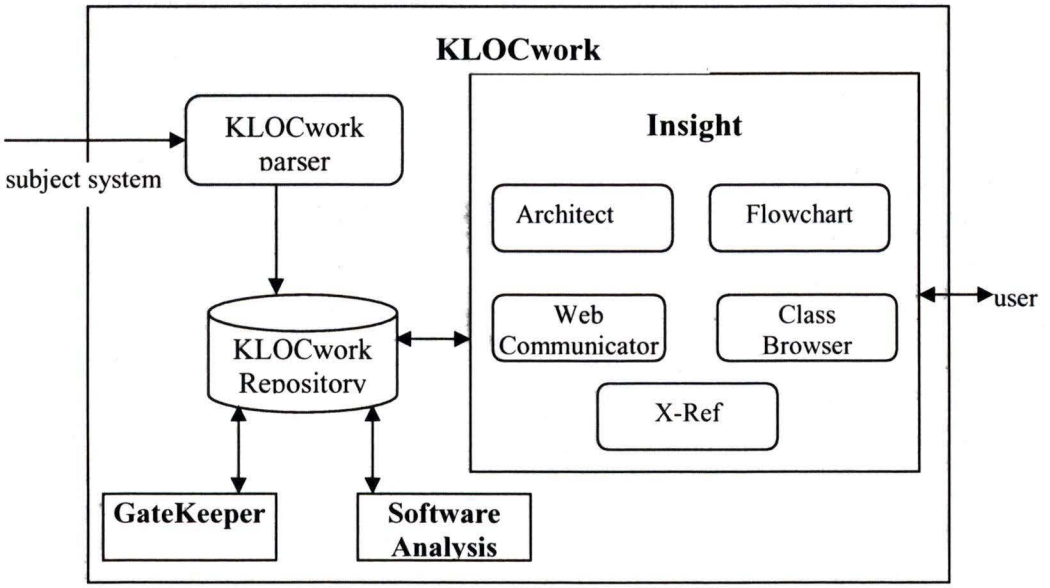


Figure 3.1. KLOCwork Architecture

3.2.2.3 KLOCwork Database

KLOCwork requires MySQL 3.23.25 to build its database. The KLOCwork Database has two major domains, the domain of entities and relations and the domain of diagrams and connectors. For our study, we needed the domain of entities and relations, as it represents the semantics of the source code. Regarding entities and relations, we used the tables defined in Table 3.1., 3.2. and 3.3.:

Field Name	Description
eid	The unique numeric identifier of the entity
kid	Kind of the entity
name	The name of the entity
file_eid	The eid of the file
lineno	The line number where the entity starts
parent_eid	Logical parent eid of the current eid
Son_did	Reference to the diagram

Table 3.1. KLOCwork entity table description

Field Name	Description
rid	The unique identifier of the relation
from_eid	The eid of the source of the relation
to_eid	The eid of the target of the relation
file_eid	The eid of the file
lineno	The line number where the entity starts

Table 3.2. KLOCwork relation table description

Field Name	Description
kid	The unique identifier of the kind
descry	The text string, describing the kind
tag	Semantic descriptor of the kind

Table 3.3. KLOCwork kind table description

In addition to the entity, relation and kind tables, we also used the metrics table for calculating TNOS and NIM measures. Each metrics table contains the fields: eid, mid (identifier of the metric), and value (value of the metric).

3.2.3 Implementation for Deriving Metrics

Our metric analysis program was written in Java to derive metrics from subsystems. Tcl/Tk, a scripting language, is often selected for metrics programs due to its facility with integrating and coordinating a set of existing components or applications. However, we chose Java, a system programming language, due to the need for good execution speed to enable the frequent scanning of big datasets, and our need for sophistication to deal with the complex calculations involved in indirect coupling and cohesion metrics in addition to personal experience. Of course, there are even faster languages than Java, e.g., C++. However, we preferred Java because of its portability and type safety.

3.3 Hypotheses and Questions

We posed the following hypotheses:

H-size: The bigger NIM and TNOS a class has, the more likely it has changes (perfective/adaptive) and bugs (corrective).

H-directCoupling: The greater the value of a class's direct coupling metric, the more likely the class has changes (perfective/adaptive) and bugs (corrective).

H-indirectCoupling: The greater the value of a class's indirect coupling metric a class has, the more likely the class has changes (perfective/adaptive) and bugs (corrective).

We also posed the following questions that we wished to answer with our empirical study:

Q_1: Which metrics have significant effects on maintainability characteristics?

Q_2: How much variability in maintainability characteristics can be explained by variability in metrics?

3.4 Analyzing Data

Analyzing data has three main steps: analyzing the evolution of metrics during the development time, analyzing the relationships among metrics, and analyzing the relationships between metrics and maintainability characteristics.

3.4.1 Analyzing the Evolution of Metrics during the Software Development

All selected classes in both systems were analyzed for the time between the class was first created and April 2002 in three-month intervals. Therefore we had eighteen sets of metric values for each subsystem. All metric values were normalized for [0,5]. Furthermore their average was taken for each set after we combined the two subsystems regarding their time intervals. This enabled us to see how the selected metrics changed during the software development, and to capture any common development characteristic that existed. If the average value was near either 0 or 5 any time during the development of software, it indicated a common development characteristic.

3.4.2 Analyzing the Relationships among Metrics

To understand the relationships between metrics, we created correlation matrixes. Although there are a total of 38 different sets of metrics data collected

between January 1998 and April 2002, we have calculated metrics correlations from a smaller time frame, between January 1999 and April 2001 to capture the biggest number of classes to analyze and to have maintainability characteristics collected for at least one year period (at least between April 2001 and April 2002), since all classes originated at different times.

Minitab Data Analyzer tool [33] was used to create correlation matrixes. Each correlation matrix has a correlation coefficient(r) value for each possible metric pair and can range from –1 and +1. An r value of +1 represents perfect positive correlation, while an r value of –1 represents perfect negative correlation.

3.4.3 Analyzing the Relationships between Metrics and Maintainability Characteristics

The purpose of this step is determining the measures, which are important to predict the maintainability of a class. The significant measures were determined with only one snapshot (an isolated observation for a given time) of the systems, and then their validation was evaluated using other snapshots. In addition to correlation matrixes, regression models were used for determining the measures, which are important to predict the maintainability of a class.

3.4.3.1 Regression Analysis

The first step of the regression analysis [34], univariate regression analysis technique, was applied to all selected measures to see the relationship between each individual measure, and the perfective/adaptive and corrective maintainability

characteristics. The second step, multivariate regression analysis technique, was then applied to the combination of measures, which have a significant relationship with the perfective/adaptive and corrective maintainability characteristics.

3.4.3.1.1 Univariate Regression Analysis

One way of determining the association between two measures is to calculate a correlation coefficient [34]. However, a high value of a correlation coefficient does not always mean a perfect relationship exists between the variables. Consequently we also analyzed the p-values of the relationships between metrics and maintenance characteristics to evaluate the strength of the evidence. The smaller the p-value, the more statistically significant the result.

3.4.3.1.2 Multivariate Regression Analysis

Multiple regression is used to account for the variance in an interval dependent, based on linear combinations of independent variables [34]. Multiple regression can establish that a set of independent variables explains a proportion of the variance in a dependent variable at a significant level (significance test of R^2), and can establish the relative predictive importance of the independent variables (comparing beta weights). One can test the significance of the difference of two R^2 values to determine if adding an independent variable to the model helps significantly.

The multiple regression equation takes the form $y = b_1x_1 + b_2x_2 + ... + b_nx_n + c$. The b's are the regression coefficients, representing the amount the dependent variable y changes when the independent variable changes 1 unit. The c is the constant, where the regression line intercepts the y-axis, representing the value of the dependent y when all the independent variables are set to 0. The standardized version of the b coefficients are the beta weights, and the ratio of the beta coefficients is the ratio of the relative predictive power of the independent variables. Associated with multiple regression is R^2, multiple correlation. This is the percent of variance in the dependent variable explained collectively by all of the independent variables.

R^2 (R-squared), also called multiple correlation or the coefficient of multiple determination, is the percent of the variance in the dependent explained uniquely or jointly by the independents. R-squared can also be interpreted as the proportionate reduction in error in estimating the dependent when knowing the independents. In other words, R^2 reflects the number of errors made when using the regression model to guess the value of the dependent, in ratio to the total errors made when using only the dependent's mean as the basis for estimating all cases. Mathematically, $R^2 = 1 - (SSE/SST)$, where SSE = error sum of squares = $SUM((Y_i - EstY_i)^2)$, where Y_i is the actual value of Y for the ith case and $EstY_i$ is the regression prediction for the ith case; and where SST = total sum of squares = $SUM((Yi - MeanY)^2)$. In summary, R^2 is 1 minus the regression error as a percent of total error and will be 0 when regression error is as large as it would be if one simply guessed the mean for all cases of Y.

Adjusted R-Square is an adjustment for the fact that when one has a large number of independents, it is possible that R^2 will become artificially larger. This is an approximately unbiased estimate of the population R-Square. At the extreme, when there are as many independents as observations in the sample, R^2 will always be 1.0. The adjustment to the formula arbitrarily lowers R^2, as the number of independents increases. When used for the case of a few independents, R^2 and adjusted R^2 will be close. When there are a great many independents, the adjusted R^2 may be noticeably lower. The greater the number of independents, the more the researcher is expected to report the adjusted coefficient.

3.4.3.2 Stepwise and Best Subset Regression

If there are a large number of possible explanatory variables, stepwise regression can be used to select which of these variables plays a significant part in the model. Since there are a number of measures to be examined, this technique can select the measures that affect the maintainability characteristics.

Stepwise multiple regression, also called statistical regression, is a way of computing regression in stages [33]. In stage one, the independent best correlated with the dependent is included in the equation. In the second stage, the remaining independent with the highest partial correlation with the dependent, controlling for the first independent, is entered. This process is repeated until the addition of a remaining independent does not increase R-squared by a significant amount (or until all variables are entered, of course). This mode of operation is called forward

stepwise regression in Minitab. Alternatively, the process can work backward, starting with all variables and eliminating independents one at a time until the elimination of one makes a significant difference in R-squared. The correct use of both forward and backward methods can ensure that as many combinations of variables as possible have been tried.

Minitab develops this stepwise regression technique one step further. The technique, called best subset, is based on stepwise regression but it gives us the user-defined number of best selections instead of just the top one. Therefore, we used this technique to avoid a biased selection of measures.

3.5 Summary

In this chapter, the process of designing our empirical study, in particular the relationships between the set of object-oriented measures and maintainability characteristics, was explained. In addition, we presented the hypotheses and questions, which we wanted to evaluate at the end of the empirical study.

The University of Paderborn provided the subject subsystems. Their metrics were derived by a program written in Java and with the help of a round trip engineering tool, KLOCwork InSight.

The statistical techniques used to analyze the data were presented. We have decided the system should be analyzed in three steps:

1- Analyzing the evolution of metrics for each class in the subsystems to capture common development characteristics if such characteristics exist.

2- Analyzing the relationships between metrics through the use of correlation matrixes.

3- Analyzing the relationships between metrics and maintainability characteristics by using regression analysis technique. The first step of the regression analysis, univariate regression analysis technique, was supported through correlation matrixes. The second step of the regression analysis, the multivariate regression analysis technique, was supported through stepwise and best subset regression techniques.

Moreover, the relationship between maintainability characteristics perfective/adaptive and corrective versus preventive was analyzed to see whether a significant correlation exists. If it exists, preventive maintenance characteristics also affect perfective/adaptive and corrective maintenance characteristics and should be used along with the selected metrics to determine relationships between the measures and the maintainability characteristics.

Chapter 4

Case Study Results

This chapter presents the results of the empirical study whose design was introduced in Chapter 3.

4.1 Analyzing the Evolution of the Metrics during the Software Development

As mentioned in Chapter 3, the purpose of this study was to capture the common software development behaviors in both of selected subsystems. Metrics were calculated for each class in the subsystems throughout the development process and their average has been calculated after their values were normalized. If the average values of metrics change between near 0 and near 5, it shows common system development behaviors. How the metrics change during the development can be seen in the following line graphs.

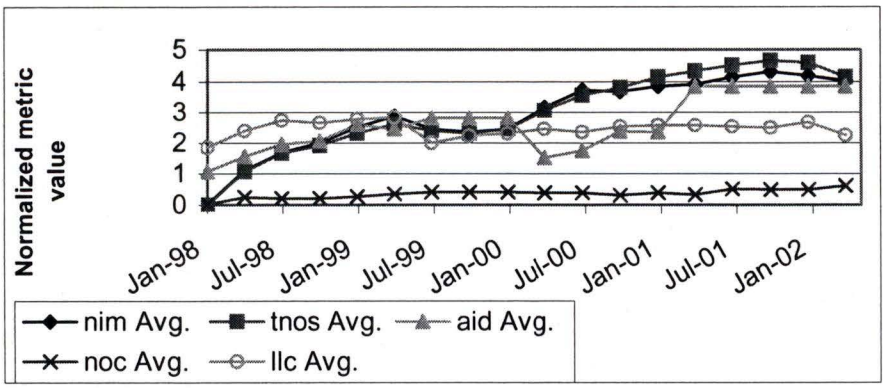


Figure 4.1. Line diagram for the evolution of nim, tnos, aid, noc, and llc metrics during the development of software

As seen in the Figure 4.1, both systems show a steady increase in the total number of statements (TNOS) and the number of instance methods (NIM) metrics. In addition there is no common development behavior among classes regarding the average inheritance depth tree (AID), number of children (NOC), and cohesion (LLC) metrics.

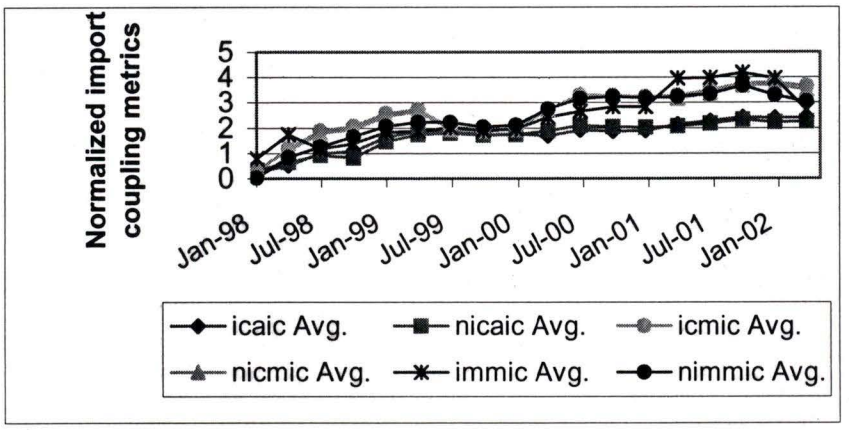


Figure 4.2. Line diagram for the evolution of icaic, icmic, immic, nicaic, nicmic, and nimmic metrics during the development of software

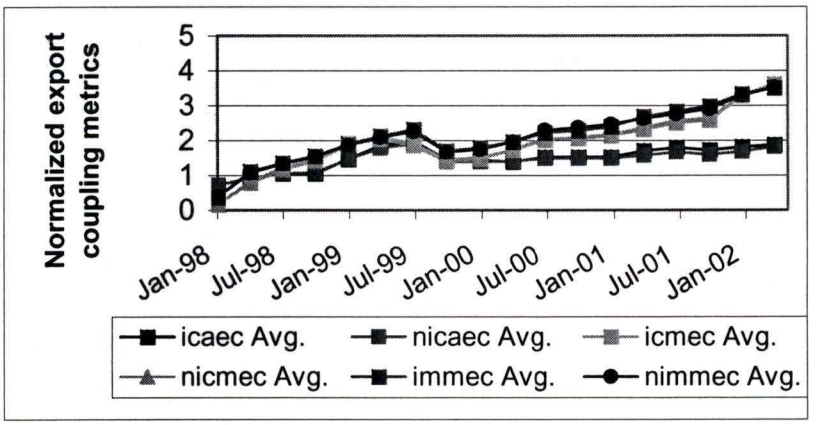


Figure 4.3. Line diagram for the evolution of icaec, icmec, immec, nicaec, nicmec, and nimmec metrics during the development of software

Figure 4.2 and 4.3 exhibits the similar development behavior for the two systems. Regardless of inheritance and non-inheritance relations, the evolution of the number of class-attribute import/export coupling interactions does not indicate a common software development behavior while the number of class-method and method-method import/export coupling interactions are increasing slightly during the development and is thus a common behavior during the development of systems.

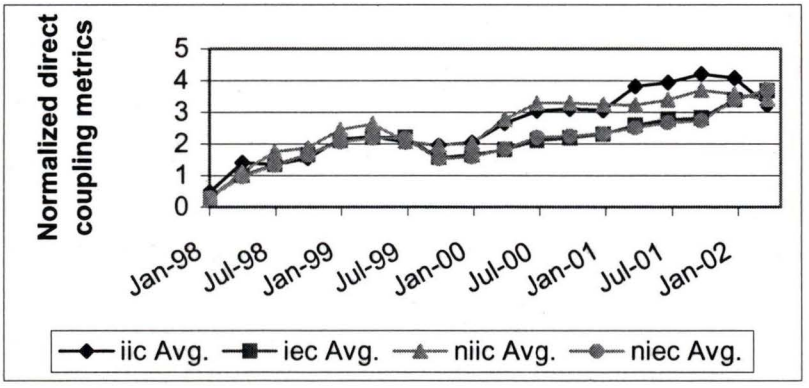


Figure 4.4. Line diagram for the evolution of iic, iec, niic, and niec metrics during the development of software

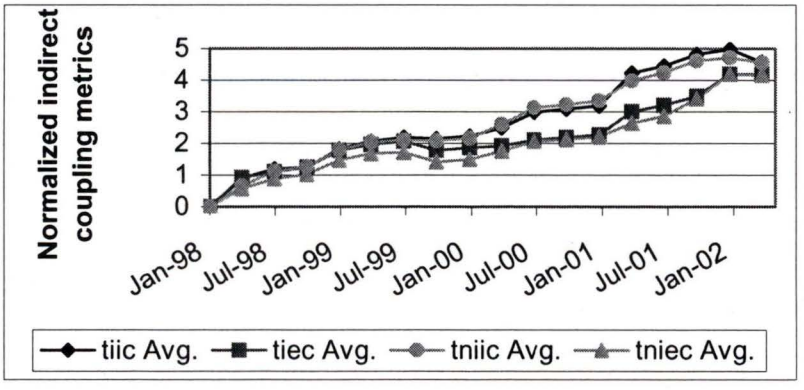


Figure 4.5. Line diagram for the evolution of tiic, tiec, tniic, and tniec metrics during the development of software

As seen in Figures 4.4 and 4.5, the interactions between classes are increasing during development from the class and the system point of view. This was expected since we have already pointed out that increasing class-method and method-method interactions are a common software development behavior.

Likewise in Figures 4.6 and 4.7, common software development behavior is preserved regarding direct and indirect coupling metrics, counting classes coupled. This behavior not only emphasizes that increasing direct and indirect coupling relationships is a common software development behavior, but it also emphasizes that one of the reasons of the increasing direct and indirect number of interactions for a class (as seen in Figures 4.4 and 4.5) is the increasing total number of interacting classes.

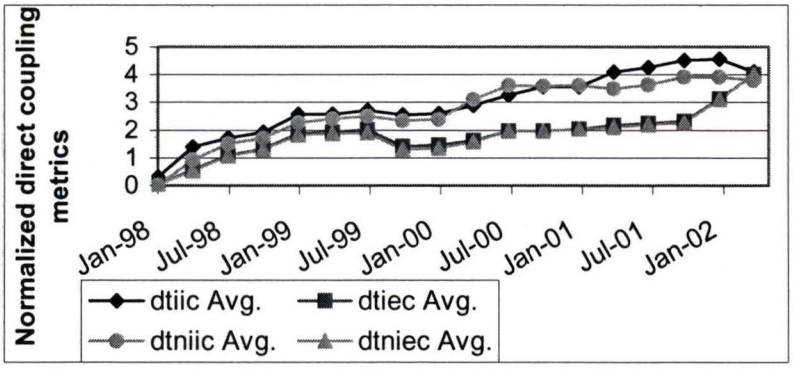


Figure 4.6. Line diagram for the evolution of dtiic, dtiec, dtniic, and dtniec metrics during the development of software

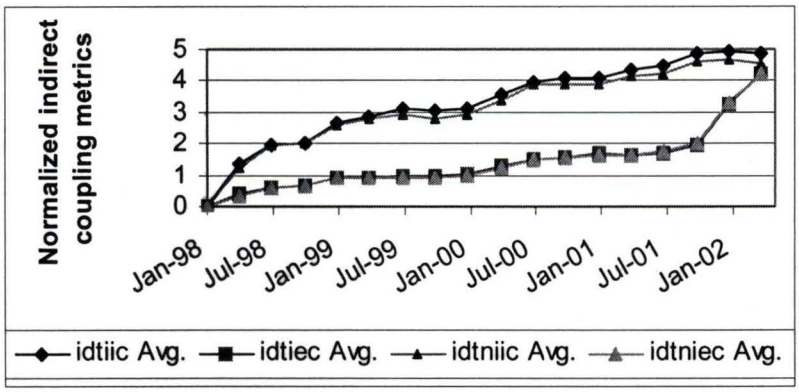


Figure 4.7. Line diagram for the evolution of idtiic, idtiec, idtniic and idtniec metrics during the development of software

4.2 Analyzing the Relationships among Metrics

Both *dobs* and *uml* system's metrics were gathered for the period between January 1999 and April 2001. To understand the relationships between metrics, correlation matrixes were created and analyzed every three months, twenty matrixes in total. Any value between 0.70 and 1.00 was accepted as a strong positive correlation, while any value between –1.00 and –0.70 was accepted as a strong negative correlation. This enabled us to derive the following results from a correlation table (see Appendix A) derived from the arithmetic average of all the correlation matrixes.

- There is a strong positive relationship among TNOS, NIM, and all direct coupling metrics.
- There are strong positive relationships among export coupling metrics and among import coupling metrics.

- In contrast to our expectation about indirect coupling metrics, there are only weak correlations between indirect coupling and other metrics. The reason for this is discussed in the next section.

4.3 Analyzing the Relationships between Metrics and Maintainability Characteristics

4.3.1 Selecting Individual Metrics as a Significant Predictor for Maintainability Characteristics

As discussed in chapter 3, the correlation matrix and univariate analysis technique were used to select the measures, which have a significant impact on maintainability characteristics. The following table shows typical logs and how their categorizations are made under maintainability characteristics:

Logs	p/a	corr	pre
revision 1.260 date: 2000/06/08 13:12:38; author: felix; state: Exp; lines: +39 -17 some changes	2		
revision 1.231 date: 2000/04/20 11:20:30; author: wag25; state: Exp; lines: +4 -31 Comments removed.			1
revision 1.170 date: 1999/05/12 09:00:27; author: mtt; state: Exp; lines: +5 -2 fixed two bugs introduced yesterday		2	
revision 1.163 date: 1999/02/11 10:51:57; author: nierej; state: Exp; lines: +14 -3 Bug fix in killing all tokens		1	
revision 1.214 date: 2000/03/23 14:47:52; author: creckord; state: Exp; lines: +126 -21 added propertyChange support to all uml classes	1		

Table 4.1. Typical log entries with their categorizations under maintainability characteristics (derived from cvs log entries for UMLClass.java file)

The previous correlation matrix, used to analyze relationships among metrics, was analyzed to investigate the effects of metrics on maintainability (Appendix A). It was found that there is a strong relationship between perfective/adaptive maintenance and corrective maintenance characteristics. Consequently, any metric having a strong correlation with perfective/adaptive maintenance characteristic also has a strong correlation with corrective maintenance characteristic, and vice versa. The relationship between preventive maintenance and other maintenance characteristics was also examined since preventive maintenance characteristic is a future predictor and can decrease the impact of metrics on other maintainability characteristics.

In addition to the correlation matrix, p-values were derived from univariate analysis (Appendix B). For the p-value, the confidence level was set to be at least 95%. The metrics were selected as significant if their p-values, calculated for each metric corresponding to each maintainability characteristic, were greater than or equal to the accepted confidence level and their correlation with maintainability characteristics was strong. Thus, we determined:

- TNOS and NIM metrics are good predictors with a 99.9% confidence level for maintainability characteristics.
- Export coupling metrics are not a good predictor for maintainability characteristics.
- No significant relation was found between indirect coupling metrics and maintainability characteristics: Indirect coupling metrics were thought to be a good predictor for maintainability characteristics since direct coupling metrics do

not have the ability to distinguish classes as seen in Figure 4.8. This result was initially quite puzzling. As seen in the example, direct coupling metrics do not have the ability to distinguish between class A and class X while indirect coupling metrics do. Consequently indirect coupling metrics were thought to be a good predictor. Unfortunately this situation is only valid for the systems with only basic interactions. In today's complex software systems, indirect coupling metrics do not distinguish between most classes due to interactions between the classes being circular. This kind of situation is illustrated in Figure 4.9. As seen in the figure, the indirect importing coupling metrics eventually become the same for A, C, and F classes. Because these circular interactions exist in most complex systems, in contrast to our belief at the beginning of the study, indirect coupling metrics do not represent good predictions for maintainability characteristics.

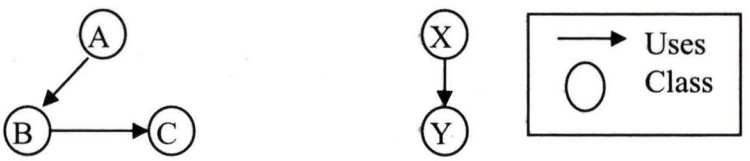


Figure 4.8. Basic example showing the importance of indirect coupling metrics

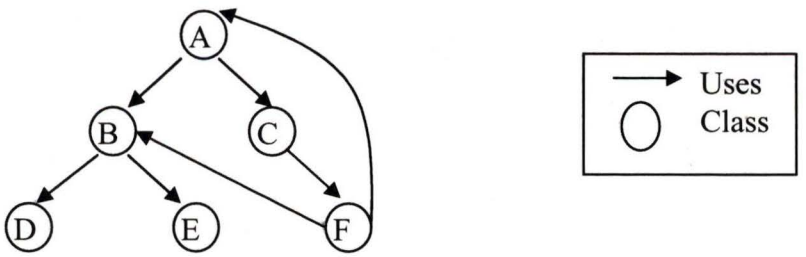


Figure 4.9. Example showing a circular kind of interaction

- A strong relation was found between direct import coupling metrics and maintainability characteristics with a 97.5 - 99.9% confidence level.

Based on the above results, the metrics that have been used to predict maintainability in our multivariate analysis were chosen to be: TNOS, NIM, ICMIC, NICMIC, IMMIC, NIMMIC, IIC, NIIC, and DTNIIC.

4.3.2 Using a Collection of Metrics to Predict Maintainability

Choosing the predictors among selected measures is perhaps the most important step in our empirical study. Although selected measures are highly correlated with the maintainability characteristics that we want to predict, they may lead to wrong results if they are used in an inappropriate combination. Thus, the importance of correlation between measures should not be neglected when using measures in combination. When the selected measures are observed, there is clearly a strong correlation between non-inheritance — inheritance metric and TNOS — NIM metric pairs. To make the best selections, these metrics are evaluated as both combined and separated among other selected metrics. For this purpose, R-square and R-square (adjusted) values, derived from the multiple regression model, were compared. As seen in Table 4.1, calculating the inheritance metrics in addition to the non-inheritance metrics did not increase the power of the prediction model for corrective and perfective/adaptive maintenance. Therefore, NICMIC, NIMMIC, and NIIC measures were selected over ICMIC, IMMIC, and IIC measures.

versus corr. - p/a	R-square	R-square(adj)
iic, niic, icmic, nicmic, immic, nimmic	82.40% - 85.00%	79.00% - 82.10%
iic, icmic, immic	65.30% - 68.20%	62.30% - 65.40%
niic, nicmic, nimmic	80.70% - 84.00%	79.00% - 82.60%

Table 4.2. R-square and R-square (adjusted) values to compare the advantage of different set of measures. (subject system: uml 04/1999)

This lowered the number of predictors to six by eliminating the inheritance metrics. As a next step, the best subset regression model was used for selecting the best combination of the remaining metrics. The reason for not using this technique from the beginning was to make our selection easier. The "best subset" technique is limited to showing a maximum of the five best predictors and this might make it difficult to select metrics because of the strong correlation among inheritance and non-inheritance metrics.

The collection of the predictor measures is TNOS, NIMMIC, NICMIC, and NIIC, according to the following Minitab best subset output for corrective maintenance characteristics. As seen in Table 4.3, C-p values were also given. When the C-p value is small, bias of the regression model is small. In addition to the C-p value, selection was made based on R-square (adjusted) and the definition of good metrics. Moreover, 86.1% R-Square and 84.4% R-Square (adjusted) values were derived from multiple regression models when these candidate metrics were applied to predict perfective/adaptive maintenance.

Vars	R-Sq	R-Sq(adj)	C-p	S	tnos	nimic	nimmic	nimic	dtnic
1	66.6	65.6	30.8	0.58614	X				
1	63.0	62.0	37.7	0.61654		X			
1	62.2	61.2	39.3	0.62314			X		
1	60.0	58.9	43.5	0.64081				X	
2	69.5	67.8	27.2	0.56785		X		X	
2	69.1	67.3	27.9	0.57149	X		X		
2	67.8	65.9	30.5	0.58371	X			X	
2	66.8	64.9	32.3	0.59215	X	X			
3	80.7	79.0	7.4	0.45839			X	X	X
3	75.6	73.5	17.2	0.51488	X			X	X
3	74.5	72.2	19.5	0.52696		X		X	X
3	73.3	70.9	21.8	0.53902	X		X	X	
4	*83.2*	*81.2*	*4.5*	*0.43357*	*X*		*X*	*X*	*X*
4	82.3	80.1	6.4	0.44572		X	X	X	X
4	81.1	78.8	8.7	0.46036			X	X	X X
4	76.3	73.4	18.0	0.51555	X	X		X	X
5	83.9	81.4	5.1	0.43095	X		X	X	X X
5	83.2	80.6	6.5	0.44022	X	X	X	X	X
5	83.2	80.6	6.5	0.44040		X	X	X	X X
5	76.4	72.7	19.8	0.52248	X	X		X	X X
6	84.0	80.9	7.0	0.43680	X	X	X	X	X X

Table 4.3. Best subset regression model to select the most significant combination of measures (for corrective maintenance). (subject system: uml 04/1999)

4.4 Testing Metrics as a Predictor of Maintainability Characteristics

As mentioned before, we have also analyzed the role of preventive maintenance characteristics on perfective/adaptive and corrective maintenance characteristic. The role of preventive maintenance is usually forgotten while designing empirical studies and so misleading results are produced. Unfortunately, we also had to ignore the preventive maintenance characteristic since preventive maintenance was performed in ad-hoc manner during the development of both systems and no significant correlation was found with perfective/adaptive and corrective maintenance characteristics as seen in Table 4.4. While we were not able to add the effects of performing preventive maintenance to the predictive model, it may be assumed that the created model may be unstable. However, not counting the role

of preventive maintenance will only slightly affect the predictor since little preventive maintenance was performed during the development of *dobs* and *uml*. Moreover, as discussed in Chapter 2, we reconfirmed the distribution of maintainability characteristics with Figure 4.10. The way how these maintainability characteristics were acquired is shown in Figure 4.11. Therefore, clearly corrective maintenance (the fixing of defects) is always performed by changing the code in which perfective/adaptive maintainability characteristics are performed most.

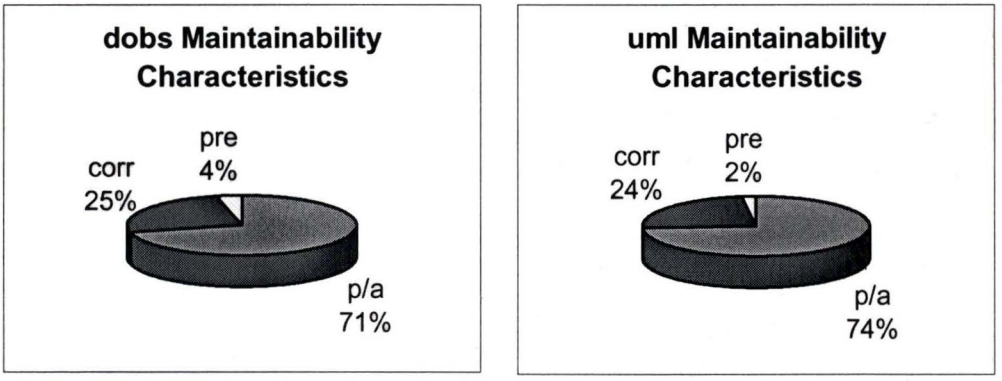


Figure 4.10. Distribution of the maintainability characteristics in dobs (533 records) and uml (3120 records) systems

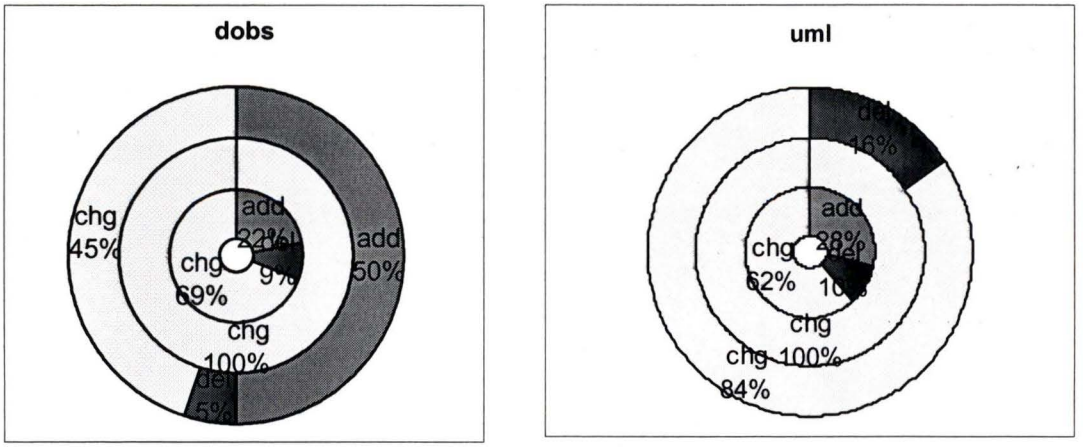


Figure 4.11. Doughnut chart showing how maintainability characteristics have been acquired during the development (inner circle: p/a, middle circle: corr., outer circle: pre.)

	p/a	corr	
preventive maintenance	0.51	0.511	dobs
	0.575	0.499	
	0.558	0.485	
	0.471	0.36	
	0.512	0.456	
	0.5	0.493	
	0.588	0.557	
	0.495	0.447	
	0.517	0.447	
	0.532	0.436	
	0.557	0.459	uml
	0.607	0.504	
	0.606	0.536	
	0.621	0.566	
	0.592	0.526	
	0.569	0.555	
	0.552	0.602	
	0.581	0.5	
	0.609	0.472	
	0.63	0.588	
Avg.	0.5591	0.49995	

Table 4.4. Correlation table, preventive versus perfective/adaptive and corrective maintenance characteristics (for the period between 01/1999 and 04/2001)

To test the validation of selected measures, they were applied to each snapshot of the systems and analyzed to investigate to what level they capture the class' perfective/adaptive and corrective maintenance characteristics.

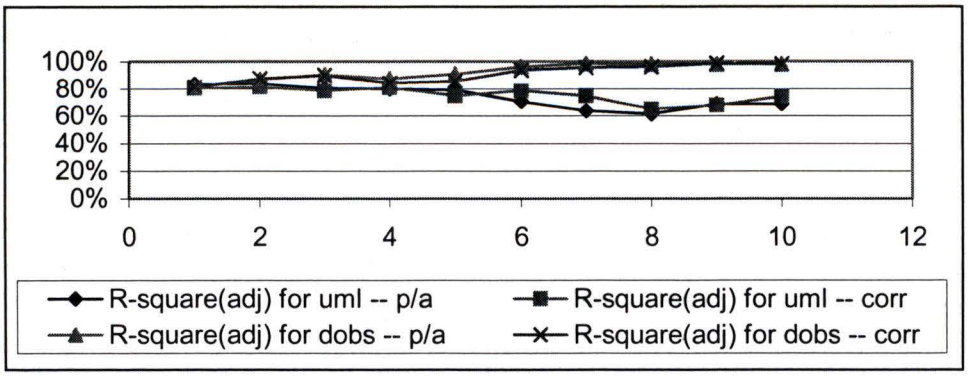


Figure 4.12. Line diagram showing at what level selected metrics capture maintainability characteristics

According to the above figure, TNOS, NICMIC, NIMMIC, and NIIC are still very good predictors even when applied with other snapshots of the systems. In the chart we see the R-square adjusted values versus the perfective/adaptive and corrective maintenance characteristics for both the *uml* and *dobs* system. R-square adjusted values are between 61.60% and 99.70%. There is a good correlation for predicting perfective/adaptive and corrective maintenance characteristics while R-square adjusted values are slightly different between the two systems. When we searched for the reason for this difference, we found that some reused classes were added to big *uml* system. This made the predictions lower than expected since reused classes are more stable and they did not require as many changes and bug fixes as other implemented classes. So, we conclude that the selected measures are good predictors when we ignore added reused classes, and they are able to capture at least 80% of the maintainability characteristics of the classes investigated.

This result is promising for future research on maintainability although ideally we would like to predict characteristics of the classes with 100% confidence. However, this is probably impossible for real world systems since we are trying to capture the maintainability with objective predictors without taking into account subjectivity. In more detail, subjectivity can be explained as experiences of the different software developers under different working conditions.

4.5 Summary

In this chapter, both the *uml* and *dobs* systems were analyzed to select a set of predictor metrics for maintainability characteristics. Therefore, questions Q_1 and Q_2 posed in Chapter 3, are now answered. We found that the suite TNOS, NICMIC, NIMMIC, and NIIC represents the best predictor for maintainability characteristics. These significant measures are able to capture 80% of the class's maintainability characteristics. The reasons of not capturing more of the class's maintainability characteristics are:

- Psychological situations and the experience of the software developers were ignored.
- The role of preventive maintenance characteristics could not be counted as an effect to other maintainability characteristics.
- Indirect coupling metrics could not be used efficiently and effectively.

Moreover, the hypotheses posed in Chapter 3 were evaluated as follows:

H-size: has been validated

H-directCoupling: has been validated

H-indirectCoupling: Could not be shown. The reason of why existing indirect coupling measures do not work in practice has been discussed. Despite not being able to show how the evolution of indirect coupling metrics are affecting maintainability, they may be used to compare the maintainability of different systems in future empirical studies. To compare classes of the same system, it may be calculated until some degree of the step to avoid having the same metric values

because of the circular interactions among classes as seen in the following figure. For instance, while the TNIIC for class A and for class C are the same for our indirect coupling metric calculation, their TNIIC is different if we stop the calculation at some pre-defined level. For instance, if we run the algorithm for two steps TNIIC for class A will be ab+ac+bd+be+cf interactions and TNIIC for class C will be cf+fa+fb interactions.

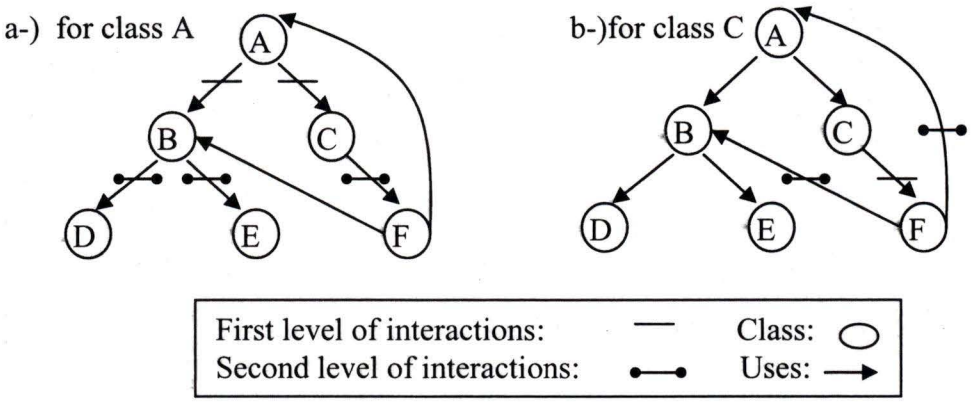


Figure 4.13. Example for indirect coupling metric calculation (two level)

Chapter 5

Conclusions

5.1 Summary

In this thesis, the selection of the best collection of measures on maintainability characteristics was made. To accomplish this, a great variety of measures were selected for empirical evaluation. While the focus was on the effects of coupling measures rather than size, cohesion, and inheritance measures, we attempted to calculate coupling measures using different aspects. These aspects are: export, import, direct, indirect coupling, counting or not counting the interactions with inherited classes, and counting the number of classes that the class interacted with or counting the number of interactions of the class.

For our empirical study we used two subsystems developed at the University of Paderborn. It was found that some coupling metrics and size metrics are good predictors for maintainability characteristics. To make the best collection of significant metrics, different statistical methods were used. The measures TNOS, NICMIC, NIMMIC, and NIIC were then chosen as the best predictor measures for maintainability characteristics.

However in this study, we also found that indirect coupling measures and export-coupling measures were not very useful as direct import coupling measures, despite the initial expectations.

5.2 Contributions

The following contributions have been made as a result of this thesis:

- We have analyzed different measures proposed in the object-oriented software literature and upper-selected with thirty-three measures including the measures that we have extended and created from scratch. We argued that these measures were the most efficient and effective measures to examine software maintainability according to our theoretical research.
- We have empirically validated that TNOS, NICMIC, NIMMIC, and NIIC measures are good predictors for maintainability characteristics.
- Our implemented metric analysis engine has been integrated with KLOCwork InSight in terms of a first prototype. A more full-featured and usable implementation is future work.
- We have proposed new indirect coupling measure in Chapter 4, since our indirect coupling measures were not efficient to predict class behaviors especially in same systems.
- We have analyzed log files of two different sized systems and categorized log entries under three maintainability characteristics: perfective/adaptive, corrective and preventive. We also stored the information whether adding, deleting, and

changing process have caused those characteristics. Since all the information is kept in excel files, we encourage researchers to use them for their empirical studies.

5.3 Related and Future work

A vast body of research has been performed on the subject of measures and maintainability. The software maintenance literature contains many ideas and techniques on how to capture the maintainability of software. However many of them are only defined theoretically and are not evaluated in empirical studies.

One of the most recent studies showing relations between measures and maintainability is proposed by Muthanna *et al.* [35]. Similar to our empirical study, this paper first investigates measures for predicting maintainability and performs correlation analysis between the measures and maintainability data. In contrast to our empirical study, measures are for traditional software systems, and the maintainability data is subjective since it is only collected from the results of questionnaires given to the software developers. Muthanna *et al.* categorize classes in software systems according to their maintainability data into maintainability categories low, medium, and high. Then, they predict maintainability of modules through the use of their regression model. Eventually, their model was able to capture the maintainability of 15 out of the 21 classes they studied. Their approach, categorizing maintainability data, is a good way to distinguish classes in software systems. However, we do not think categorizing into three groups is precise enough for most software systems. The

evaluation of predicting maintainability should be done by using the maintainability data categorized under more groups.

In Yu's thesis, ten internal object-oriented product measures have been empirically validated [40]. Since they have analyzed and tested the metrics suite for the same single industrial software system, their prediction formula is questionable and more empirical studies are needed. From the maintainability point of view, more empirical studies should be done for the other maintainability characteristics in addition to fault-proneness (corrective maintenance characteristic). An important result is that they found coupling measures within and across inheritance hierarchies show significantly different effects on fault-proneness. We confirmed their result, as our non-inheritance coupling measures are more effective to predict maintainability characteristics than our inheritance coupling measures.

Emam *et al.* have empirically shown that the size of a class has a confounding effect on the validity of object-oriented metrics for measuring fault-proneness [36]. They have also stated the hypothesis that this was true for measuring other characteristics of object-oriented study. Our study confirms this hypothesis for maintainability characteristics.

Another interesting approach to assessing the quality of Java software has been presented by Patenaude *et al.* [37]. The authors show how facts extracted by the DATRIX^{tm} parser can be used to assess maintainability aspects of object-oriented software such as clone-proneness. The authors' argument is based on the common understanding that the existence of software clones has negative effects on the

maintainability of a system. Thus there is no need to use historical data to validate the metrics suite proposed.

Recently, there have been studies using other properties to assess the maintainability of software system. For instance, Alagar *et al.* propose some measures for architectural slicing. They claim that architectural slices can be used to reduce the effort in examining software by allowing a maintainer to focus attention on the set of objects to be affected by the changes only [38]. However, their theoretical approach has yet to be tested. Aggarwal et al. propose another uncommon approach for predicting maintainability using a fuzzy approach [4].

The above studies are promising for the future to predict maintainability of software systems. However, empirical studies must be done, and then their good features should be combined for future analysis.

Some future topics were identified as a continuation of the work presented in this thesis:

- Systä *et al.* have combined the use of object-oriented metrics and graphical visualization techniques for assessing and communicating software characteristics [39]. We plan to integrate our metrics engine with the KLOCwork visualization tools.
- As the significant metrics suite presented in this thesis, some other empirical studies should be done to develop universal prediction model.
- The predictive model should try to include the role of preventive maintenance characteristics in empirical studies for predicting maintenance characteristics.

- The presented indirect coupling measures (Chapter 2) can be evaluated in empirical studies to compare classes of different systems.
- The presented indirect coupling measures can be calculated by limiting their calculations until some level of degree to avoid circular interactions as discussed in Chapter 4.

Bibliography

[1] Schach, S., "Software Engineering," Irwin Publishers, 1996.

[2] Sommerville, I., "Software Engineering," Addison Wesley, 1996.

[3] Tahvildari, L.; Gregory, R. and Kontogiannis, K., "An approach for measuring software evolution using source code features," In Proceedings of 6th Asia Pacific Software Engineering Conference (APSEC '99), pages 10-17, 1999.

[4] Aggarwal, K. K.; Singh, Y. and Chhabra, J. K., "An integrated measure of software maintainability," In Proceedings of Reliability and Maintainability Symposium, pages 235-241, 2002.

[5] McCabe, T. J., "A Complexity Measure," IEEE Transactions on Software Engineering, Volume: SE-2, No. 4, pages 308-320, 1976.

[6] Halstead, M. H., "Elements of Software Engineering," Elsevier, New York, 1977.

[7] Rocacher, D., "Metrics Definitions for Smalltalk," Project ESPRIT 1257, MUSE WP9A, 1988.

[8] Voegele, J., "Programming Language Comparison" http://www.jvoegele.com/software/langcomp.html

[9] Fenton, N., "Software Metrics — A Rigorous Approach," Chapman & Hall, London, pages 42, 1991.

[10] Chidamber, S. R. and Kemerer, C.F., "A metrics suite for object oriented design," IEEE Transactions on Software Engineering, Volume: 20, Issue: 6, pages 476-493, June 1994.

[11] Briand, L. C.; Daly, J. W. and Wüst, J. K., "A Unified Framework for Coupling Measurement in Object-Oriented Systems," IEEE Transactions on Software Engineering, Volume: 25, No. 1, pages 91-121, January 1999.

[12] ISO, "Information Technology — Software Product evaluation — Quality Characteristics and Guidelines for their Use," International Standard ISO/IEC 9126, Geneva, 1995.

[13] Moore, J. W., "An integrated collection of software engineering standards," IEEE Software, Volume: 16, Issue: 6, pages 51-57, November/December 1999.

[14] Slaughter, S. A. and Banker R. D., "A study of the effects of software development practices on software maintenance effort," In Proceedings of Software Maintenance, pages 197-205, 4-8 November 1996.

[15] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," report IEEE Std 610.12-1990, 1990.

[16] Swanson, E. B., "The Dimensions of Maintenance," In Proceedings of 2nd International Conference on Software Engineering, IEEE, pages 492-497, 1976.

[17] Pressman, Roger, S., "A Practitioner's Approach 3rd Ed.," Software Engineering, McGraw-Hill Book Co., 1992.

[18] Booch, G., "Object-Oriented Analysis and Design with Applications," 2nd Edition, Benjamin/ Cummings Pub. Co.Inc. California., 1994.

[19] Leintz, B. P. and Swanson E. B., "Software Maintenance Management," Addison Wesley, 1980.

[20] Sousa, M. J. C. and Moreira H. M., "A Survey on Software Maintenance Process," International Conference on Software Maintenance (ICSM98), IEEE Press, 1998.

[21] Lorenz, M. and Kidd, J., "Object-Oriented software metrics," New Jersey, Prentice Hall,1994.

[22] Chidamber, S. and Kemerer, C., "A Metrics Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, pages 476-492, June 1994.

[23] Henderson-Sellers, B., "Software Metrics," Prentice Hall, Hemel Hempstaed, U.K., 1996.

[24] Henderson-Sellers, B. and Constantine, L. L., "Coupling and Cohesion: towards a valid suite of object-oriented metrics," Object-Oriented Systems 3(3), pages 143-158, 1996.

[25] Rosenberg, L. H., SATC. "Applying and Interperting Object oriented Metrics," presented at the Software Technology Conference, Utah, April 1998.

[26] Bieman, J. M. and Kang, B., "Cohesion and reuse in an object-oriented system," In Proceedings of the Symposium on Software Reusability (SSR'95), pages 259-262,

Appendix A

Correlation Matrix for Measures, and Maintainability Characteristics

Average corr.	nim	tnos	aid	noc	llc	icaic	nicaic	icaec	nicaec	icmic	nicmic
Tnos	0.90										
Aid	-0.17	0.02									
Noc	0.20	0.00	-0.30								
Llc	0.12	0.20	-0.26	-0.17							
Icaic	0.79	0.74	-0.11	-0.07	0.17						
Nicaic	0.79	0.74	-0.12	-0.06	0.16	0.97					
Icaec	0.33	0.17	-0.23	0.67	-0.07	0.07	0.07				
Nicaec	0.24	0.11	-0.17	0.48	-0.03	0.09	0.10	0.90			
Icmic	0.83	0.91	-0.02	0.01	0.28	0.78	0.78	0.21	0.19		
Nicmic	0.86	0.90	-0.08	0.02	0.28	0.82	0.83	0.22	0.19	0.99	
Icmec	0.58	0.45	-0.18	0.53	-0.02	0.42	0.41	0.65	0.65	0.47	0.49
Nicmec	0.55	0.44	-0.14	0.39	0.01	0.44	0.43	0.58	0.64	0.47	0.49
Immic	0.60	0.85	0.27	-0.10	0.21	0.53	0.53	0.11	0.10	0.85	0.80
Nimmic	0.73	0.87	0.07	-0.07	0.28	0.69	0.69	0.15	0.13	0.93	0.91
Immec	0.55	0.33	-0.34	0.71	-0.06	0.26	0.26	0.78	0.63	0.32	0.35
nimmec	0.58	0.39	-0.31	0.33	0.08	0.44	0.43	0.53	0.60	0.42	0.45
Niic	0.82	0.90	-0.01	-0.03	0.28	0.80	0.80	0.18	0.16	0.98	0.98
Niec	0.56	0.42	-0.22	0.39	0.04	0.43	0.42	0.61	0.69	0.45	0.47
Iic	0.74	0.92	0.14	-0.06	0.25	0.68	0.68	0.15	0.14	0.96	0.93
Iec	0.58	0.39	-0.28	0.67	-0.05	0.33	0.33	0.79	0.69	0.39	0.42
Tniic	0.32	0.33	-0.22	0.16	0.33	0.26	0.26	0.27	0.26	0.45	0.44
Tniec	0.23	0.26	-0.14	-0.05	0.29	0.18	0.17	0.17	0.18	0.24	0.24
Tiic	0.10	0.05	0.13	0.06	-0.05	0.11	0.11	0.10	0.10	0.22	0.22
Tiec	0.27	0.31	-0.28	0.12	0.45	0.19	0.19	0.20	0.20	0.30	0.30
Dtniic	0.84	0.85	-0.01	0.04	0.20	0.73	0.70	0.24	0.21	0.91	0.92
Dtniec	0.70	0.63	-0.30	0.23	0.25	0.54	0.52	0.36	0.37	0.59	0.61
Dtiic	0.81	0.85	0.13	-0.01	0.16	0.70	0.68	0.20	0.18	0.91	0.90
Dtiec	0.65	0.49	-0.40	0.71	0.09	0.36	0.36	0.64	0.48	0.46	0.48
Idtniic	0.32	0.33	-0.22	0.16	0.32	0.25	0.25	0.26	0.26	0.44	0.44
Idtniec	0.20	0.22	-0.09	-0.07	0.23	0.15	0.15	0.14	0.16	0.20	0.19
Idtiic	0.07	0.01	0.15	0.05	-0.10	0.08	0.09	0.08	0.08	0.18	0.18
Idtiec	0.22	0.28	-0.23	0.13	0.42	0.14	0.14	0.22	0.22	0.27	0.26
p/a	0.88	0.89	-0.07	0.12	0.16	0.67	0.67	0.24	0.15	0.81	0.80
Corr	0.83	0.88	-0.04	0.08	0.14	0.65	0.65	0.23	0.15	0.76	0.75
Total	0.88	0.90	-0.06	0.11	0.16	0.67	0.67	0.24	0.15	0.80	0.80

Average corr.	icmec	nicmec	immic	nimmic	immec	nimmec	niic	niec	iic	iec
Tnos										
Aid										
Noc										
Llc										
Icaic										
Nicaic										
Icaec										
Nicaec										
Icmic										
Nicmic										
Icmec										
Nicmec	0.98									
Immic	0.30	0.31								
Nimmic	0.32	0.34	0.93							
Immec	0.84	0.76	0.10	0.18						
Nimmec	0.85	0.87	0.17	0.28	0.80					
Niic	0.42	0.43	0.88	0.97	0.27	0.38				
Niec	0.95	0.97	0.25	0.31	0.81	0.95	0.40			
Iic	0.38	0.39	0.97	0.96	0.20	0.29	0.96	0.35		
Iec	0.93	0.87	0.19	0.25	0.97	0.83	0.34	0.89	0.29	
Tniic	0.33	0.32	0.32	0.38	0.24	0.29	0.42	0.32	0.40	0.29
Tniec	0.23	0.26	0.19	0.22	0.21	0.29	0.23	0.28	0.22	0.22
Tiic	0.03	0.03	0.17	0.19	-0.02	-0.07	0.21	0.00	0.20	0.01
Tiec	0.29	0.30	0.21	0.27	0.27	0.35	0.28	0.32	0.26	0.28
Dtniic	0.47	0.47	0.75	0.85	0.38	0.47	0.90	0.47	0.86	0.43
Dtniec	0.75	0.77	0.37	0.47	0.61	0.75	0.56	0.77	0.49	0.67
Dtiic	0.42	0.42	0.79	0.87	0.30	0.40	0.90	0.42	0.88	0.36
Dtiec	0.80	0.71	0.23	0.33	0.85	0.67	0.41	0.71	0.34	0.86
Idtniic	0.32	0.31	0.32	0.38	0.24	0.29	0.41	0.31	0.39	0.28
Idtniec	0.19	0.23	0.16	0.18	0.17	0.24	0.19	0.23	0.18	0.18
Idtiic	-0.01	0.00	0.14	0.15	-0.05	-0.10	0.17	-0.03	0.17	-0.02
Idtiec	0.30	0.31	0.21	0.24	0.24	0.31	0.25	0.31	0.24	0.27
p/a	0.40	0.36	0.70	0.78	0.40	0.40	0.81	0.38	0.78	0.42
Corr	0.40	0.37	0.70	0.74	0.37	0.38	0.77	0.38	0.76	0.40
Total	0.40	0.37	0.71	0.78	0.39	0.40	0.81	0.39	0.78	0.42

Average corr.	tniic	tniec	tiic	tiec	dtniic	dtniec	dtiic	dtiec	idtniic	idtniec	idtiic	idtiec
Tnos												
Aid												
Noc												
Llc												
Icaic												
Nicaic												
Icaec												
Nicaec												
Icmic												
Nicmic												
Icmec												
Nicmec												
Immic												
Nimmic												
Immec												
Nimmec												
Niic												
Niec												
Iic												
Iec												
Tniic												
Tniec	0.19											
Tiic	0.36	-0.48										
Tiec	0.25	1.00	-0.48									
Dtniic	0.43	0.26	0.19	0.29								
Dtniec	0.43	0.42	0.15	0.51	0.61							
Dtiic	0.40	0.22	0.47	0.32	0.98	0.53						
Dtiec	0.41	0.26	0.15	0.47	0.46	0.91	0.40					
Idtniic	1.00	0.17	0.28	0.17	0.42	0.45	0.54	0.52				
Idtniec	0.13	0.98	-0.26	0.74	0.22	0.27	0.14	0.35	0.13			
Idtiic	0.33	-0.56	0.96	-0.56	0.15	-0.08	0.21	-0.04	0.35	-0.54		
Idtiec	0.25	0.97	-0.46	0.97	0.26	0.49	0.22	0.42	0.23	0.96	-0.54	
p/a	0.31	0.14	0.33	0.34	0.82	0.63	0.61	0.33	0.42	0.14	0.12	0.16
Corr	0.26	0.21	0.20	0.40	0.73	0.62	0.53	0.34	0.53	0.21	-0.04	0.22
Total	0.30	0.16	0.30	0.36	0.80	0.64	0.60	0.33	0.58	0.40	0.08	0.18

Average corr.	p/a	corr
Corr	0.94	
Total	1.00	0.97

Appendix B

p-Values for Metrics, and Maintainability Characteristics

p-Value Avg.	nim	tnos	aid	noc	llc	icaic	nicaic
p/a	0.001	0.000	0.664	0.404	0.372	0.014	0.014
corr	0.005	0.000	0.619	0.499	0.416	0.025	0.025
Total	0.001	0.000	0.642	0.420	0.370	0.014	0.015

p-Value Avg.	icaec	nicaec	icmic	nicmic	icmec	nicmec	immic
p/a	0.450	0.502	0.003	0.003	0.453	0.447	0.025
corr	0.442	0.500	0.013	0.015	0.423	0.407	0.034
Total	0.447	0.499	0.004	0.004	0.454	0.444	0.025

p-Value Avg.	nimmic	immec	nimmec	niic	niec	iic	iec
p/a	0.003	0.439	0.312	0.001	0.394	0.005	0.447
corr	0.016	0.399	0.301	0.012	0.360	0.012	0.414
Total	0.005	0.430	0.308	0.002	0.385	0.005	0.442

p-Value Avg.	tniic	tniec	tiic	tiec	dtniic	dtniec	dtiic
p/a	0.411	0.587	0.536	0.484	0.002	0.115	0.002
corr	0.475	0.413	0.661	0.357	0.010	0.113	0.010
Total	0.426	0.531	0.549	0.435	0.003	0.112	0.002

p-Value Avg.	dtiec	idtniic	idtniec	idtiic	idtiec
p/a	0.221	0.420	0.611	0.554	0.454
corr	0.205	0.492	0.430	0.689	0.318
Total	0.215	0.437	0.553	0.576	0.403

VITA

Surname: Dagpinar Given Names: Melis

Place of Birth: Istanbul, Turkey

Educational Institutions Attended:

University of Victoria	2000 to 2003
University of Istanbul	1995 to 1999

Degrees Awarded:

B.Eng	University of Istanbul	1999

Honours and Awards:

Grad Teaching & Research Fellowships	2000 to 2002

UNIVERSITY OF VICTORIA PARTIAL COPYRIGHT LICENSE

I hereby grant the right to lend my thesis to users of the University of Victoria Library, and to make single copies only for such users or in response to a request from the Library of any other university, or similar institution, on its behalf or for one of its users. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by me or a member of the University designated by me. It is understood that copying or publication of this thesis for financial gain by the University of Victoria shall not be allowed without my written permission.

Title of Thesis/Dissertation:

Predicting Software Maintainability by Using Object-Oriented Metrics

Author 

Melis Dagpinar

August 01, 2003