Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison

Conference Paper · December 2003

DOI: 10.1109/WCRE.2003.1287246 · Source: IEEE Xplore

Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison

Melis Dagpinar and Jens H. Jahnke
*net*lab, Department of Computer Science*
University of Victoria, Canada B.C.
email: [melisd,jens]@cs.uvic.ca

Abstract

A large number of metrics have been proposed for measuring properties of object-oriented software such as size, inheritance, cohesion and coupling. We have been investigating which of these object-oriented metrics can be used as significant predictors for the maintainability of software. For this purpose, we have designed and conducted an empirical study based on historical data collected from the maintenance history of a medium-sized object-oriented system. Unlike most related studies, indirect coupling has also been taken into account in our work in order to evaluate its impact. Our study uses the maintenance history of two software systems as evidence base for linking software quality attributes to metrics suggested for object-oriented software. Our results indicate that size and import direct coupling metrics are significant predictors for measuring maintainability of classes while inheritance, cohesion, and indirect/export coupling measures are not.

1. Introduction

Object-oriented design and programming is the dominant development paradigm for software systems today. With the growing complexity and size of object-oriented systems, the ability to reason about quality attributes based on automatically computable measures has become increasingly important. Several software quality attributes such as functionality, portability, usability, efficiency, and maintainability have been defined by various individuals and standardization bodies, e.g., [1]. Maintainability is a particularly interesting quality attribute as it has been recognized that software maintenance activities account for the largest cost in today's software development [19]. On the other hand, maintainability might also be among the quality attributes most difficult to estimate, because this inherently involves making predictions about future activities. The IEEE Standard Glossary of Software Engineering defines maintainability as

> " 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." [11]

Despite the fact that software maintenance is an expensive and challenging task, it is often poorly managed. One reason for poor management is the lack of proven measures for software maintainability [2]. By having the proven measures, project managers can estimate the time that will be spent to evolve the system or show the validation of their system regarding maintainability, one of the important software quality characteristics [1]. Our research tries to improve this situation by contributing empirical evidence about the usefulness of object-oriented metrics for predicting maintainability of object-oriented software.

In a collaborative project involving the Universities of Victoria, Alberta and Paderborn, and an industrial partner, KLOCwork Solutions Inc., we have collected and analyzed historical data about the maintenance history of two sample software systems over a period of three years. We have surveyed, categorized and evaluated existing object-oriented metrics with respect to their significance for predicting maintainability of object-oriented software. Unlike most related studies, indirect coupling has also been taken into account in our work in order to evaluate its impact. Our results indicate that size and import direct coupling metrics are significant predictors for measuring maintainability of classes while inheritance, cohesion, and indirect/export coupling measures are not.

This paper is structured as follows. The next section surveys object-oriented metrics. Section III describes the design of the conducted study, while Section IV reports its results. We discuss related work in Section V and close with conclusions in Section VI.

This research has been funded in part by KLOCwork Solutions Inc., the National Science and Research Council (NSERC) and the Consortium for Software Engineering Research (CSER).

2. Object-Oriented Metrics

Since their first appearance in the 1970's, automatically computable metrics have become an important tool for assessing attributes of software and software-related activities.

Fenton has roughly categorized software metrics as

- process metrics for measuring characteristics of software development processes,
- product metrics for assessing software products such as components, procedures and programs, and
- resource metrics for measuring characteristics of software-related resources such as hardware and personnel [10].

Fenton furthermore distinguishes between internal (examining only the software) and external (examining the software in a environment) metrics. Using this terminology, this paper focuses on internal product metrics for object-oriented software. Rocacher was one of the first to suggest that traditional metrics could not be considered sufficient for assessing object-oriented systems because of their ignorance about important object-oriented concepts like data-centric encapsulation, inheritance, polymorphism etc. [17]. 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 measure the design structures and/or data structures independently.

The metrics that we have used for our empirical study can be categorized within four groups, namely size metrics, inheritance metrics, cohesion metrics and coupling metrics.

2.1. Size Metrics

Size metrics are perhaps the most frequently used metrics in practice. There exists rich empirical evidence for correlating the size of a software system with its fault-proneness [9]. The simplest and most commonly used size metric is lines of code (LOC). Is drawback of using LOC for measuring object-oriented systems is that it is not aware of the key concept of information hiding (encapsulation). Moreover, LOC measures highly depend on coding style of programmers. Therefore, we have considered in our study two object-oriented size metrics for measuring interface size and code size of classes. NIM (Number of Instance Methods) counts the number of instance methods in a class, i.e., all public, protected and private methods defined in the interface of instances of a given class [12]. TNOS (Total Number Of Statements) is an extension of Lorenz and Kidd's metric to count the number of statements in a method for assessing entire classes [12].

2.2. Inheritance Metrics

We included inheritance metrics in our study because of the often-cited fragile base class problem, which essentially characterizes the fragility of subclass implementations with respect to changes in their base classes [13]. Some of the most commonly applied inheritance metrics are DIT (Depth of Inheritance Tree) and NOC (Number of Children) both included in the CK metric suite proposed by Chidamber and Kemerer [6]. NOC for a class is defined by calculating the number of child classes one level below the selected class. DIT is not well defined in case of multiple inheritance. Therefore, Henderson-Sellers has proposed the AID metric (Average Inheritance Depth), which defines the AID of a class as the average AID of its parents (or zero, if there are no parent classes). We adopted NOC and AID for this study.

2.3. Cohesion Metrics

High cohesion has traditionally been a desirable property of classes in object-oriented software systems. Several metrics have been proposed to assess the degree of cohesion. Most of them are derived from Chidamber and Kemerer's LCOM (Lack of Cohesion in Methods) metric, which, in its original form, counts the number of pairs of methods in a class using no attribute in common [6]. Since this metric highly correlates with size metrics that count the number of methods in a class, LCOM was redefined to counting the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. It has been pointed out that LCOM lacks sufficient discriminative power to distinguish between classes with dramatically different cohesion [18]. Moreover, indirect attribute dependencies are not taken into account in LCOM.

Therefore, we have chosen Bieman and Kang's LCC (Loose Class Cohesion) metric, which considers direct as well as indirect attribute dependencies [4]. An attribute (instance variable of a class) 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 relative number of pairs of methods that are directly or indirectly connected. Bieman and Kang propose three ways to calculate LCC, namely

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.

We have chosen the second alternative for our study.

2.4. Coupling Metrics

Software engineering best practices promote low coupling between components in order to decrease inter-dependencies and facilitate evolution. Currently available evidence (and common sense) suggest that coupling metrics as good predictors for the maintainability of components in object-oriented systems. However, there are many different types of coupling, such as inheritance vs. non-inheritance coupling, import vs. export coupling, direct vs. indirect coupling, etc. and insufficient empirical evidence exists about their significance for predicting maintainability.

The first argument about inheritance vs. non-inheritance coupling has been started by Chidamber and Kemerers CBO (coupling between object classes) definition [6]. 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. Many other suggestions for measuring coupling have been made. For instance, the metrics presented by Rajaraman and Lyu [22], distinguishes 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). Notably, Rajaraman and Lyu do not evaluate the significance of CNIC and CIC separately in their empirical study, but they consider only the sum of these measures (CC).

Few empirical studies differentiate between export coupling and import coupling. Export coupling focuses on interactions of a class, in which it is used by other classes, while import coupling considers interactions of that class using the functionality of other classes. We have considered separate export and import coupling measures since we believe that there could be a distinction between a server and client class from the perspective of maintenance.

Likewise, we could find few empirical studies validating the significance of indirect coupling (vs. direct coupling) for assessing quality attributes such as maintainability. We wanted to investigate and compare the significance of indirect coupling metrics and included eight such measures in our study. Briand et al. categorize three types of interactions [5]:

1. 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.

2. 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 of class d, or class c is the return type of method m_d.

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

The measures used in our empirical study are mainly based on measures defined by Briand et al., with the following modifications:

- A separation has been made in order 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 the number of interactions between classes, the number of classes that the class interacts with is also counted.

The coupling measures considered in our study are listed in the following tables. We refer to [7] for a more detailed coverage of each of these metrics.

Table 1. Direct coupling measures counting the number of interactions

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 = ICAIC+ICMIC+IMMIC
NIIC	Non-inheritance import coupling. NIIC = NICAIC+NICMIC+NIMMIC
IEC	Inheritance export coupling. IEC = ICAEC+ICMEC+IMMEC
NIEC	Non-inheritance export coupling. NIEC = NICAEC+NICMEC+NIMMEC

Table 2. Indirect 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 3. Direct coupling measures counting the number of classes

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 4. Indirect 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.

3. Case Study Design

3.1 Selection of subject systems

The purpose of the study was to investigate the significance of various object-oriented metrics for the purpose of predicting maintainability of software. Rather than relying on subjective estimates about future maintenance effort, we wanted to use historical data about maintenance records of real-world example systems to achieve a more objective evaluation. The following requirements drove our selection of particular subject systems:

- Subject systems should have well-documented maintenance records that span at least over two years for most of their classes.
- We wanted to study subject systems of different overall size, in order to see whether our results differ with different overall systems size.
- Subject systems should have been developed by different teams.
- Subject systems should have been developed in C++ or Java. (This requirement originated from the fact that we wanted to use the KLOCwork tool suite and repository for parsing and storing facts about the software systems. At the time of starting the study, C++ and Java were the only two object-oriented languages supported by KLOCwork.)

We selected two Java systems Fujaba-UML (FUML) and Dynamic Object Browser (dobs) that have been developed at the Software Engineering Group at the University of Paderborn, Germany. FUML and dobs both belong to the public domain FUJABA software engineering tool environment (www.fujaba.de). The development of the first version of FUML was done by three graduate students in 1998. Its current version contains approximately 180 classes (April 2002). Dobs was developed in 1998 by a team of graduate students as a dynamic object-debugging tool. It contains 27 classes (April 2002). Obviously, there was a lot of fluctuation in the designs of both systems during the early development phases. We determined that no major design changes were performed between January 1999 and January 2002. Thus, we selected this three year window for our study.

3.2. Data Gathering

We planned to gather two types of data about the two systems, namely metrics data and historical data about their actual maintenance history.

3.2.1 Maintenance history. The Software Engineering Group at the University of Paderborn provided us with access to their historical software repository via their Distributed Software Development (DSD) system (www.upb.de/cs/dsd). Using DSD, we retrieved all maintenance logs in the chosen three-year interval, which were created by their internal CVS version management system. We planned to use the log messages to categorize maintenance activities according to Swanson's dimensions of maintenance, namely perfective, adaptive,

and corrective maintenance [20]. In addition, we considered preventive maintenance a fourth category as an activity to improve the future maintainability of a system [16]. An analysis of the log messages showed, however, that we could not sufficiently discriminate between adaptive and perfective maintenance activities. Therefore, we counted both types of activities into one joint category, which consequently left us with three distinguishable categories perfective/adaptive, corrective and preventive maintenance.

Table 5. Typical log entries with their categorizations according to maintenance dimensions

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		

The actual developer messages contained in the log entries were of concise nature but still descriptive enough for discriminating among the three categories. Table V shows five examples for such log entries for class UMLClass of system FUML including their categorization. Log entry 3 and 4 illustrate that the developer teams clearly marked corrective maintenance activities by words like bug and fix. After filtering corrective maintenance activities, we had to discriminate between preventive and perfective/adaptive. This process was not quite as straight forward and required some understanding of the nature of the modifications, which, at times, could only be provided by the developers in Paderborn. An example for such an ambiguous log entry is presented in the first row in Table V.

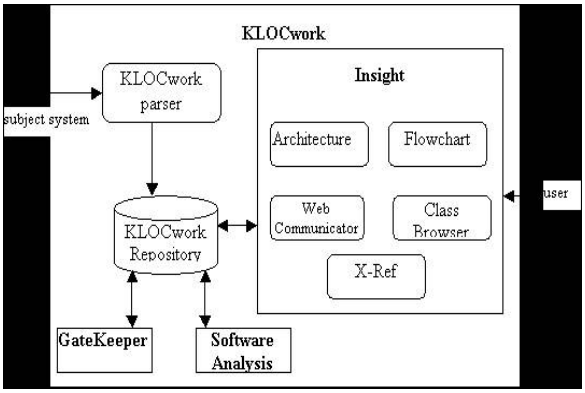


Figure 1. KLOCwork Suite Overview

3.2.2 Metrics data. In order to gather the metrics data required for our study, we repeatedly used DSD to retrieved snapshots of the source code of the two systems over time intervals of three months. We adopted and extended the KLOCwork tool suite for extracting the metrics about the system snapshots. KLOCwork already provides most if the functionality needed, including robust parsers, fact extractors and a comprehensive repository hosted by a relational database management system. Figure 1 gives a high-level overview of the main components of the KLOCwork tool suite. Metrics are already computed and used in two new KLOCwork tools called Gate Keeper and Software Analysis. However, no object-oriented metric was implemented at the time we started our study. Therefore, we had to implement all metrics except for TNOS (Total Number of Statements) and NIM (Number of Instance Methods). Still, implementing the metrics was relatively straightforward once we understood the structure of the KLOCwork fact repository. We used SQL and Java (JDBC) to calculate the metrics. The KLOCwork components shown in the box entitled Insight in Figure 1 are tightly integrated information browsers and interactive refactoring tools.

- Web Communicator: allows users to view help documentations and user comments added for specified entities.
- Architect: presents graphical models of systems' 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.

We used them to get a better understanding of the subject systems. However, this knowledge (and hence detailed knowledge about these tools) is not required for our actual study. We refer to [7] for details about these interactive tools.

4. Data Analysis and Results

4.1. Analysis of maintenance characteristics

After categorizing the log entries in the maintenance history of both systems, we found that only a small percentage of the maintenance effort was spent of preventive maintenance (2-4%). Approximately a fourth was spent a corrective activities and the largest effort was spent on adaptive/perfective maintenance (cf. Figure 2). This result has to be seen in context of the fact that both systems are research prototype implementations. Thus, the traditional emphasis in developing these systems has been on adding and adapting functionality, not on optimizing maintainability of existing functionality. We expect that the proportion of preventive maintenance activities would be higher in commercial systems than in research prototypes. However, we did not find comparable empirical studies that would allow us to validate this hypothesis. Furthermore, we confirmed that the frequency of preventive/adaptive maintenance activities per class correlates highly with the number of corrective maintenance activities. Therefore, good predictors for corrective maintainability would also be good predictors for preventive/adaptive maintainability.

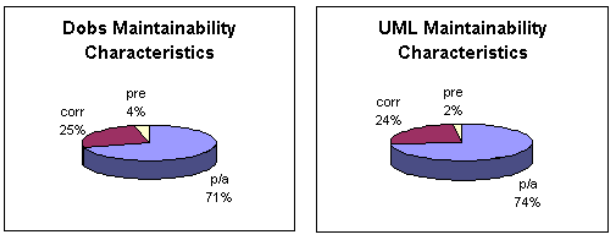


Figure 2. Distribution of the maintainability characteristics in Dobs and FUML

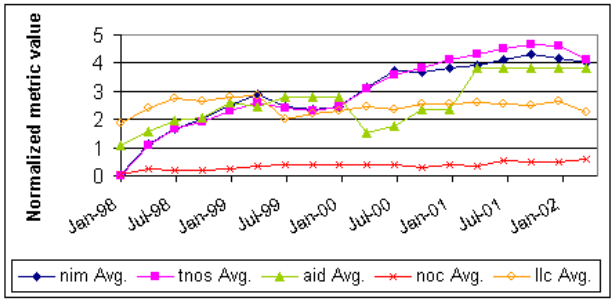


Figure 3. Evolution of average size, inheritance and cohesion metrics over duration of two years

4.2. Evolution of Metrics during Software Development

We applied all selected metrics to the classes of the two systems in three-months intervals until April 2002 and calculated the average normalized metrics values, in order to get an impression how the different object-oriented metrics evolve over the lifetime of our subject software. Figure 3 shows the evolution of size, inheritance and cohesion metrics over the selected two year development period.

The data analysis activities were driven by the general question about which metrics or suite of metrics are best suited as predictors for the maintainability of object-oriented classes. In particular, we wanted to evaluate the importance of import coupling metrics versus export coupling metrics, respectively, direct coupling metrics vs. indirect coupling metrics for this purpose. There is a steady growth in the (average) number of statements (TNOS) and the number of instance methods (NIM) per class. On the other hand, the class inheritance depth and number of child classes do not show this clear trend. In fact the number of child classes (NOC) remains more or less constant, while the average number of inheritance depth (AID) shows unsteady growth. This result suggests that evolution in our object-oriented systems is performed mainly by increasing the size of existing classes and adding functionality by sub-typing, rather than implementing new specialized classes. The selected cohesion metric LLC does not show significant loss or gain of cohesion over the study period.

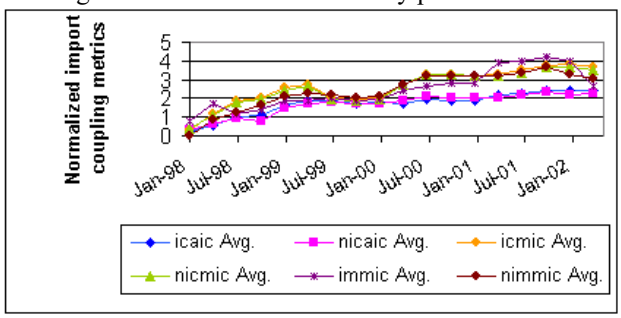


Figure 4. Evolution of direct import coupling metrics counting interactions

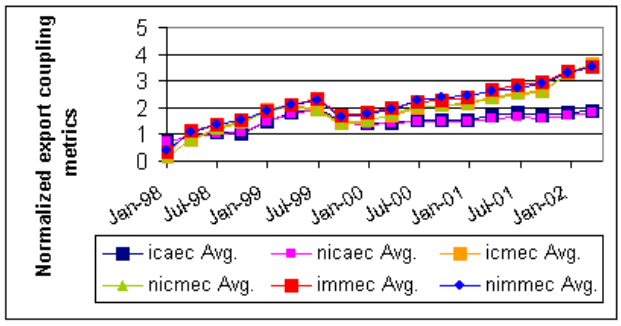


Figure 5. Evolution of direct export coupling metrics counting interactions

Figures 4 and 5 show that both graphs, computed for direct import coupling metrics and direct export coupling metrics, respectively, show almost continues growth for

the object-oriented software investigated. The same is true for indirect coupling metrics, as shown in Figure 6. The graphs produced for measuring coupling, counting the number of classes (ref. Tables III and IV) show similar growth. We did not include these graphs in this paper but they can be found in [7].

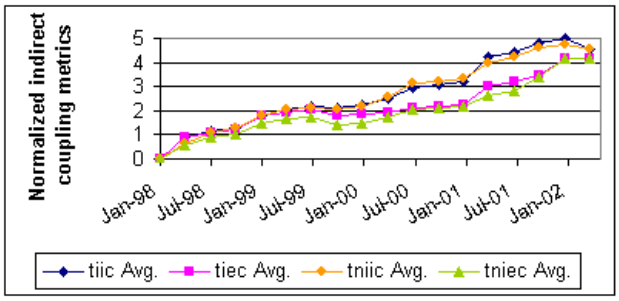


Figure 6. Evolution of indirect coupling metrics counting interactions.

4.3. Correlation among metrics

To understand the relationships between metrics, we created and analyzed correlation matrixes for every three-month interval (snapshot). Minitab data analyzer tool (www.minitab.com) was used to create correlation matrixes. Each correlation matrix has a correlation coefficient (r) for each possible metric pair and can range from -1 and +1. While the value of r, +1, represents perfect positive correlation, the value of r, -1, represents perfect negative correlation. 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. These values have been chosen as based on our subjective estimate. This enabled us to derive the followings observations derived from the arithmetic average of all the correlation matrixes.

- There is a strong positive correlation among TNOS, NIM, and all direct coupling metrics.
- There are strong positive correlations among export coupling metrics and among import coupling metrics.
- There is no strong correlation between indirect coupling and other metrics, including direct coupling metrics.

4.4. Relationship between metrics and maintainability characteristics

The purpose of this step was determining the measures, which are important to predict the maintainability of a class. The significant measures were determined with only one snapshot of the systems, and then their validation was done using the remaining snapshots. The regression model was used for determining the measures, which are important to predict the maintainability of classes.

The first step of the regression analysis, univariate regression analysis, was applied to all selected metrics in order to see the relationship between each individual measure, and the frequency of perfective/adaptive and corrective maintenance activities. The second step, multivariate regression analysis technique, was then applied to the combination of measures, which have a significant relationship with the frequency of perfective/adaptive and corrective maintenance activities.

The univariate regression analysis (Step 1) delivered the following results:

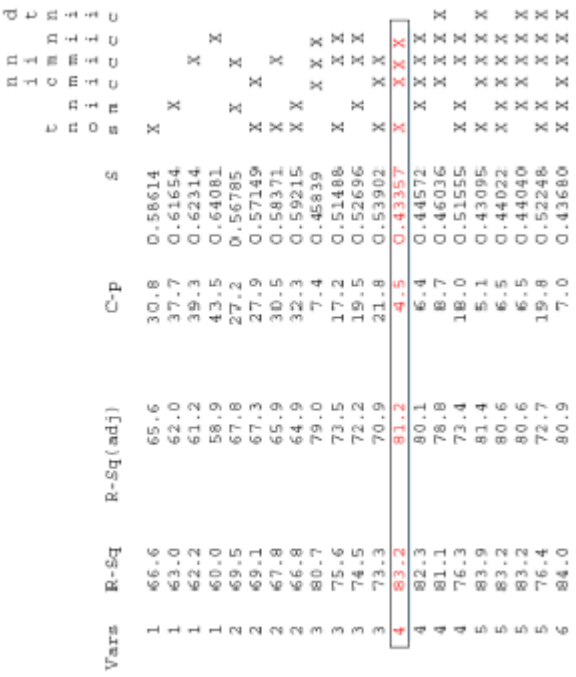
- Object-oriented size metrics TNOS and NIM metrics are good predictors for maintainability (with correlation coefficient greater or equal 0.70) with a 99.9% confidence level (p-value).
- Export coupling metrics are not a good predictor for maintainability.
- No significant relation was found between indirect coupling metrics and maintainability.
- A strong relation was found between direct import coupling metrics and maintainability characteristics with a 97.5 - 99.9% confidence level.

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%

Figure 7. R-square and R-square (adjusted) values to compare the advantage of different set of measures.

The finding that indirect coupling metrics do not significantly correlate with maintainability activities surprised us initially. We assumed that they should make good predictors for maintainability because they consider transitive dependencies and the ripple effects caused by them in case of software systems changes. Unfortunately this situation is only valid for the systems with only basic interactions. In more complex software systems, indirect coupling metrics fail to discriminate between most classes due to interactions between the classes being circular. Consequently, we selected the following metrics as candidates for our multivariate analysis: TNOS, NIM, ICMIC, NICMIC, IMMIC, NIMMIC, IIC, NIIC, and DTNIIC.

We decided to do a stepwise multiple regression analysis in order to try to further reduce the number of independent variables. In stage one, the independent best correlated with the dependent is included in the equation.



Vars	R-Sq	R-Sq(adj)	C-p	S	d i t c m n o i i i s m c c c
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	0.51408	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

Figure 8. Best subset regression model to select the most significant combination of measures

The remaining independent variables are entered in subsequent stages until the addition of a remaining independent does not increase R-squared by a significant amount. Our findings are that 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 (cf. Figure 7). Therefore, NICMIC, NIMMIC, and NIIC measures were selected over ICMIC, IMMIC, and IIC measures.

We used the best subset selection technique [8] to select our final suite of metrics best suitable as predictors of maintainability. Based on adjusted R-squared values and Mallow's C-p, we selected TNOS, NICMIC, NIMMIC and NIIC (cf. Figure 8).

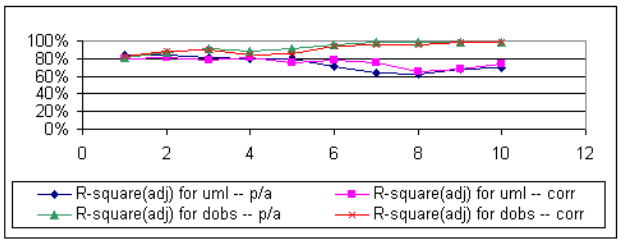


Figure 9. Line diagram showing at what level selected metrics capture maintainability characteristics

In order to test the validation of selected metrics suite, they were applied to each snapshot of the systems and analyzed to see to what level they capture the classes' perfective/adaptive and corrective maintenance characteristics. Figure 9 shows that TNOS, NICMIC, NIMMIC, and NIIC are good predictors even when applied with other snapshots of the systems. The chart in Figure 9 shows the R-square adjusted values for different snapshots of both systems with respect to perfective/adaptive maintenance and corrective maintenance characteristics, respectively. R-square adjusted values are between 61.60% and 99.70%. There is a high correlation for predicting perfective/adaptive and corrective maintenance characteristics. However, the accuracy of prediction seems to be slightly different between the two systems. We searched for the reason for this difference and noticed that several previously existing classes where reused in system FUML. We have the hypothesis that the maturity of these classes might have contributed to the difference in prediction accuracy. If this were the case, this would indicate that our metrics set could be improved by adding some time-aware maturity metrics. We intend to investigate this in our future research. When the reused classes where excluded, our metrics suite could predict at least 80% of the maintainability characteristics of the two object-oriented subject systems.

5. Related Work

Many researchers and practitioners have suggested software metrics in relationship with maintainability characteristics. However, many of these metrics are only defined theoretically and still need to be validated with empirical studies. Muthanna et al. have presented an empirical study on showing relations between software measures and maintainability at WCRE 2000 [14]. Similar to our empirical study, their paper first investigates measures for predicting maintainability and makes correlation analysis between the measures and maintainability data. In contrast to our empirical study that focuses on object-oriented metrics, Muthanna et al. consider traditional design level metrics such as function-points, data-flow, cyclomatic complexity etc. Another difference is that and the maintainability data is subjective since it was collected from the results of questionnaires given to the software developers rather than from actual data about the maintenance history. Muthanna et al. use a classification system to categorize classes into low, medium, or high maintainability, respectively. Their regression model was able to correctly predict the maintainability of 15 out of the 21 classes they studied (with respect to the collected survey data). The classification according to the three maintainability categories (low/medium/high) and the different nature of empirical data used (questionnaire vs. actual maintenance history) make it difficult to draw a firm conclusion about which metrics suite has higher predictive power. Such a

comparative study using the same data sets is a future research direction we are very interested in. From a practical point of view, our metrics suite has an advantage because it is easier and faster to compute. Note that data-flow analysis can be complex in presence of object-oriented pointers and polymorphism.

In Yu's thesis, ten internal object-oriented product measures have been empirically validated [23]. Since they have analyzed and tested the metrics suite for the same industrial software system, their prediction formula is debatable 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). One 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 [9]. They have also stated the hypothesis that this was true for measuring other characteristics of object-oriented software. Our study confirms this hypothesis for maintainability characteristics. Systa et al. have combined the use of object-oriented metrics and graphical visualization techniques for assessing and communicating software characteristics [21]. Our current work is on integrating our metrics engine with the KLOCwork visualization tools. We will build on these previous research results.

Another interesting approach to assessing the quality of Java software has been presented by Patenaude et al. [15]. The authors show how facts extracted by the DATRIX™ 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 (replicated code) has negative effects on the maintainability of a system. Thus there is no need to use historical data to validate the metrics suite proposed. Other nonstandard approaches on assessing software maintainability include measures based on program slicing [3] and fuzzy logic [2]. There is only little empirical evaluation on the advantages of these approaches and we are looking forward to further work in this direction.

6. Conclusions and Future Work

Our study provides empirical evidence that object-oriented metrics can effectively be used to predict maintainability of software systems. In contrast to other studies that use mostly subjective expert surveys to measure the predictive power of metrics, our study uses more objective data, namely the actual maintenance history of software. Our results indicate that size and import direct coupling metrics are significant predictors for measuring maintainability of classes while inheritance, cohesion, and indirect/export coupling measures are not. We have used multivariate regression analysis to select a suite of metrics that serves as best predictors for maintainability. An interesting observation that deserves further investigation is that reused classes (i.e., proven classes with higher maturity) seem to impact the accuracy of our metrics suite. Our current work is on integrating our metrics suite with visualization engines provided by the KLOCwork tool suite of our industrial partner. In the future, we intend to use our case study to compare our metrics suite with other maintainability measures suggested in literature.

References

[1] ISO/IEC 9126. "Information technology - software product evaluation- quality characteristics and guidelines for their use," 1991.

[2] Aggrawal, 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.

[3] Alagar, V. S.; Li, Q. and Ormandjieva, O. S.. "Assessment of maintainability in object-oriented software," In Proceedings of 39th International Conference on Technology and Object-Oriented Languages and Systems, pages 194-205, 2001.

[4] Bieman, J. M. and Kang, B., "Cohesion and reuse in an object-oriented system," In M. H. Samadzadeh and Man-sour K. Zand, editors, In Proceedings of the ACM SIGSOFT Symposium on Software Reusability, pages 259-262, 1995.

[5] Briand, L. C.; Daly, J. W. and Wust, J., "A unified framework for cohesion measurement in object-oriented systems," Empirical Software Engineering: An International Journal, 3(1):65-117, 1998.

[6] Chidamber, S. R. and Kemerer, C. F., "A Metric Suite for Object-Oriented Design," IEEE Transactions on Software Engineering, 20(6):476-493, 1994.

[7] Dagpinar, M., "Predicting software maintainability by using object-oriented metrics," Master's thesis, Department of Computer Science, University of Victoria, Victoria V8P3P6, B.C., Canada, 2003.

[8] Devore, J. L., "Probability and Statistics for Engineering and the Sciences," Brooks/Cole, 3rd edition, 1991.

[9] El Emam, K.; Benlarbi, S.; Goel, N. and Rai, S. N., "The confounding effect of class size on the validity of object-oriented metrics," IEEE Transactions on Software Engineering, 27(7): 630-650, 2001.

[10] Fenton, N. E., "Software Metrics: A Rigorous Approach," Chapman and Hall, London, 1992.

[11] IEEE. "IEEE Standard Glossary of Software Engineering Terminology," IEEE Std. 610.12-1990. Institute of Electrical and Electronics Engineers, 1990.

[12] Lorenz, M. and Kidd, J., "Object-Oriented Software Metrics," Prentice Hall, Englewood Cliffs, 1994.
[13] Emil Sekerinski Leonid Mikhajlov. The fragile base class problem and its impact on component systems. In Clemens Szyperski Wolfgang Weck, Jan Bosch, editor, Proceedings of the Second International Workshop on Component-Oriented Programming (WCOP'97), number 5 in TUCS General Publications, pages 59-68. Turku Centre for Computer Science, 1997.
[14] Muthanna, S.; Kontogiannis, K.; Ponnambalam, K. and Stacey, B., "A maintainability model for industrial software systems using design level metrics," In Proceedings of 7th Working Conference on Reverse Engineering, pages 248-256, 2000.
[15] Patenaude, J. F.; Merlo, E.; Dagenais, M. and Lague, B., "Extending software quality assessment techniques to java systems," In Proceedings of 7th International Workshop on Program Comprehension, Pittsburgh USA, pages 49-56, 1999.
[16] Roger S., "Pressman. Software Engineering: A Practitioner's Approach," McGraw-Hill, 1994.
[17] Rocacher, D., "Metrics definition for smalltalk," Technical report, European Union ESPRIT Research Report 1257, 1988.
[18] Rosenberg, L. H. and Hyatt, L. E., "Developing a successful metrics programme", In ESA 1996 Product Assurance Symposium and Software Product Assurance Workshop, ESTEC, Noordwijk, The Netherlands, pages 213-216, 1996.
[19] Sommerville, I. Software Engineering. Addison-Wesley, 6th edition, 2001.
[20] Swanson, E. B., "The dimensions of maintenance," In Proceedings of 2nd International Conference on Software Engineering, IEEE Computer Society Press, pages 492-497, 1976.
[21] Systä, T.; Yu, P. and Müller, H., "Analyzing java software by combining metrics and program visualization". In Proceedings of 4th European Conference on Software Maintenance and Reengineering, Zürich, Switzerland, pages 199-208, 2000.
[22] Rajaraman, C. and Lyu, M., "Reliability and Maintainability Related Software Coupling Metrics in C++ Programs," In Proceedings of the 3rd International Symposium on Software Reliability Engineering, pages 303-311, October 7-10 1992.
[23] Yu, P., "Empirical Validation of an Object-Oriented Metrics Suite ─ A Case Study," Master's thesis, Department of Computer Science, University of Victoria, V8W3P6, B.C. Canada, 2002.