FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

# ADAPTIVE SOFTWARE FAULT PREDICTION APPROACH USING OBJECT-ORIENTED METRICS

A dissertation submitted in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

by

Djuradj Babic

2012

To: Dean Amir Mirmiran
College of Engineering and Computing

This dissertation, written by Djuradj Babic, and entitled Adaptive Software Fault Prediction Approach using Object-Oriented Metrics, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this dissertation and recommend that it be approved.

Xudong He

Masoud Milani

B. M. Golam Kibria

Naphtali Rishe, Major Professor

Date of Defense: November 9, 2012

The dissertation of Djuradj Babic is approved.

Dean Amir Mirmiran
College of Engineering and Computing

Dean Lakshmi N. Reddi
University Graduate School

Florida International University, 2012

DEDICATION

Dedicated to mom.

ACKNOWLEDGMENTS

First and foremost I would like to thank my mother Gorjana Curcin and mentor Dr. Peter Clarke for their loving support and understanding throughout my academic journey. I would also like to thank my academic advisor and friend, Dr. Naphtali Rishe, and my committee members: Dr. Masoud Milani, Dr. Xudong He, and Dr. B. M. Golam Kibria for their guidance and insight into my research. Finally, I would be remiss if I did not acknowledge Ms. Olga Carbonell and the support she has so graciously provided over the years.

ABSTRACT OF DISSERTATION

ADAPTIVE SOFTWARE FAULT PREDICTION APPROACH USING OBJECT-ORIENTED METRICS

by

Djuradj Babic

Florida International University, 2012

Miami, Florida

Professor Naphtali Rishe, Major Professor

As users continually request additional functionality, software systems will continue to grow in their complexity, as well as in their susceptibility to failures. Particularly for sensitive systems requiring higher levels of reliability, faulty system modules may increase development and maintenance cost. Hence, identifying them early would support the development of reliable systems through improved scheduling and quality control. Research effort to predict software modules likely to contain faults, as a consequence, has been substantial.

Although a wide range of fault prediction models have been proposed, we remain far from having reliable tools that can be widely applied to real industrial systems. For projects with known fault histories, numerous research studies show that statistical models can provide reasonable estimates at predicting faulty modules using software metrics. However, as context-specific metrics differ from project to project, the task of predicting across projects is difficult to achieve. Prediction models obtained from one project experience are ineffective in their ability to predict fault-prone modules when applied to other projects. Hence, taking full benefit of the existing work in software development community has been substantially limited. As a step towards solving this problem, in this dissertation we propose a fault prediction approach that exploits existing prediction models, adapting them to improve their ability to predict faulty system modules across different software projects.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| CBO | Coupling Between Objects |
| CK | Chidamber and Kemerer |
| CVS | Concurrent Versions System |
| DIT | Depth of Inheritance Tree |
| IDE | Integrated Development Environment |
| JDT | Java Development Tooling |
| LCOM | Lack of Cohesion of Methods |
| LR | Logistic Regression |
| NOC | Number of Children |
| OLS | Ordinary Least Squares |
| OO | Object-Oriented |
| RFC | Response for a Class |
| SCM | Software Configuration Management System |
| SVN | Subversion |
| WMC | Weighted Methods per Class |

# CHAPTER 1

# INTRODUCTION

As users continually request additional functionality, the software systems will continue to grow in their complexity, as well as in their susceptibility to failures[1]. Particularly for sensitive systems requiring higher levels of reliability, faulty system modules[2] may increase development and maintenance cost. Furthermore, applying equal validation effort to all parts of a software system has become cost-prohibitive [12]. Hence, identifying faulty modules early would support the development of reliable systems through improved scheduling and quality control. The faulty information could provide valuable advice to improving effectiveness of resource allocation during validation activities. As numerous studies show testing software on average consumes at least 50% of its development effort [31, 41], the identification of faulty modules might have a significant cost-saving impact on software development.

A wide range of fault[3] prediction models have been proposed [10, 25, 27, 37, 45, 48, 50]. Generally, efforts have concentrated on developing statistical models that predict faulty modules a system is likely to reveal during validation activities, or within a specified time interval after its deployment. Predictive models rely on fault history data and a selection of an appropriate quality assessment model, which quantitatively evaluates some facet of system quality. System quality, such as maintainability [14] for example, is most frequently described in terms of project's complexity metrics. Numerous research studies, such as work of Basili et al. [6], Emam at al. [21], Gyimothy et al. [30], and Olague at al. [46], show that statistical models can provide reasonable estimates at predicting faulty system modules using object-oriented (OO)

[1] Any deviation of the observed system behavior from the specified behavior [64]

[2] A file, OO class, procedure, or some other system component

[3] Algorithmic cause leading to a failure [15]

metrics. However, as context-specific metrics differ from project to project, the task of predicting across projects is difficult to achieve. Prediction models obtained from one project seldom serve as predictors of fault-prone modules belonging to other projects, for which fault history data is not available [44].

The next section motivates the need for research projects in the area of software measurement and fault prediction. More specifically, it emphasizes the need for improved predictive techniques and highlights the benefits to be gained from conducting the study.

## 1.1 Motivation

Several studies have shown that software faults tend to be clustered within OO classes comprising a smaller part of the system as a whole. Ostrand et al. [49] used historical data from two large software systems with up to 17 releases to predict the files with the highest fault density in the following release. For each release, the 20% of the files with the highest predicted number of faults contained between 71% and 92% of the faults being detected. Similiary, Koru and Liu [28] showed that 80% of the changes to KOffice and Mozilla projects were centered in 20% of the classes. Given the immense weight software testing exerts on the overall software development effort outlined in the introduction, we could significantly improve effectiveness of resource allocation during validation activities by focusing testing effort on that part of the software that is likely to need it most. Research effort to predict software modules likely to contain faults, as a consequence, has been substantial.

Predictive model, alternatively referred to as a classifier, maps historical fault data of some project to its modules, and then uses their complexity metrics to predict faults in newly developed modules. We assume that if certain types of software modules were likely to fail in the past, they are also likely to do so in the future. However,

accurate predictions require a long fault history, which may not exist for the project at hand; in fact, a long fault history is something one would like to avoid altogether [44]. Hence, projects without prior fault history rely on predictive models developed from other, unrelated projects. And even though complexity metrics have been shown to correlate with fault density in a number of case studies [10, 25, 27, 37, 45, 48, 50], how do we know that chosen metrics and the predictive model are appropriate for the project at hand? Context-specific metrics differ from project to project, making the task of predicting across projects difficult to achieve. Prediction models obtained from one project seldom serve as adequate predictors of fault-prone modules belonging to other projects, for which fault history data is not available [44]. Taking full benefit of the existing work in software development community predicting faults has therefore been substantially limited.

In order to be widely adopted, suggested fault-prediction techniques should be easy-to-use and applicable across different domains. Additionally, prediction models obtained using those techniques should be simple and intuitive enough to be easily understood and interpreted by developers [28]. As new fault predictive techniques are introduced, it is necessary that researchers exchange ideas on how to utilize these techniques on arbitrary projects for which fault history data are not available. Given an arbitrary project, this includes: identifying approaches for choosing the appropriate set of complexity metrics for the project; identifying approaches for selecting and modifying existing datasets from which prediction models are developed for project at hand; and sharing the software engineering experiences of conducting such research studies. The next section concisely describes the research problem being explored in this dissertation and summarizes the major benefits to be gained from conducting the investigation.

## 1.2 Problem Statement

The problem under investigation is in the area of *software measurement and fault prediction*. More precisely, the study focuses on investigation of an adaptive fault prediction approach that exploits existing prediction techniques, adapting them to improve their ability to predict faulty system modules across different software projects. We postulate that taking advantage of the existing fault prediction techniques commercially can only be achieved through these channels of investigation. The effort is divided into the following sub-problems:

1. Devise a strategy that facilitates identifying similar projects. Project similarity implies that fault predictors obtained from one project become reasonably accurate in predicting faults in another project.

2. Formulate an approach that adapts or transforms datasets used in development of the fault-proneness prediction models in order to improve their predictive ability to identify fault-prone modules across different projects, independent of the domain used in the derivation of the predictive model.

3. Propose a high-level design methodology to support the implementation of the strategy devised in (1.) and the adaptive approaches formulated in (2.).

The results of the proposed study can lead to the: (1) discovery and understanding of deeper issues surrounding the challenges of software OO metrics as related to the fault prediction techniques; (2) development of reusable and adaptable approach to fault prediction applicable across different software projects.

## 1.3 Goals and Objectives

This section describes the goal of the research contained in this dissertation, including specific, measurable objectives that must be attained in order to satisfy that goal.

Research Goal

To assist developers identify fault prone modules for projects without prior fault history, thereby supporting the development of reliable systems and lowering the cost of software development and maintenance.

The stated research goal seeks to support the practical applicability of existing predictive techniques on projects for which the prior fault history data is not known. Without history data, rather than testing effort being distributed on the entire software project, using the techniques in this dissertation, developers can generate lists of modules likely to contain faults. This reduces the scope of software that needs to be examined, and allows more efficient resource allocation during validation and maintenance activities. The following objective and evaluation criterion will be used to measure the extent to which the research goal has been accomplished:

Objective: Allocating software validation effort on the part of the system identified by our approach shall produce better results than allocating the same effort on the part that has been selected by chance, or identified by randomly selected raw-data model.

Evaluation Criteria - Given a software system comprised of individual class modules and its fault history data summarizing each module's fault count, and the three classifiers independent of the system's domain, $P$, $P_{Rand}$ (for binary and multiple logistic regressions), and $P_{Chance}$ such that:

$P$ is a prediction model developed by our adaptive approach to identify class modules most likely to contain faults,

$P_{Rand}$ is a randomly selected raw-data fault-proneness prediction model, and

$P_{Chance}$ is a model predicting fault-prone class modules by chance,

then the predictive accuracy[4] of our model $P$ will be greater than the accuracy of $P_{Rand}$ or $P_{Chance}$, when applied to our given system.

## 1.4 Proposed Solution

To address the research problem defined in Section 1.2, we suggest an informed selection approach which identifies similar, but unrelated projects, and a general transformation function that adapts their metrics datasets in order to develop calibrated fault-proneness prediction model usable across different software systems.

We identify a set of OO metrics to be used as fault predictors by empirically validating several OO metrics that were shown to be good fault predictors in the past studies on a set of selected target applications using statistical correlation analysis. Applications serve as both our training[5] and testing datasets. Their selection is restricted to open-source applications with existing fault histories accessible through mining software repositories.

For each identified metric, along with the projects' fault history data, we generate a repository of *training datasets*, a necessary component used in the development of fault-proneness prediction models. Each pair of metric measures and fault history data within repository include a metric dataset with its fault histories from which the model can be derived.

Exploiting statistical variance analysis techniques, for a given arbitrary project, our adaptive approach to fault-proneness prediction first considers all metrics and history data pairs within the repository to identify the pair whose metric distribution most closely resembles the given project. Following the selection process, our adaptation approach utilizes general transformation function on both sets of metrics, training

[4] As measured by *F-Score*, the harmonic mean of precision and recall defined in Section 4.6

[5] A set of data used to discover potentially predictive relationships and used in the regression for development of prediction models

dataset and the given project dataset – the result is the calibrated fault-proneness prediction model with optimized predictive abilities for the given project. Finally, the proposed solution is supported with a high-level design methodology for an expandable fault prediction framework that supports practical use of fault prediction techniques in commercial setting.

## 1.5 Summary of Contributions

This dissertation establishes the following novel contributions in the area of software measurement and fault prediction:

1. Creation of an adaptive and reusable approach to building fault-proneness prediction models based on a set of OO complexity metrics and its available fault history data, applicable across system projects, and particularly to arbitrary system projects for which fault history data is not available. This includes inferential statistics and regression-based tools, and methods to assess the applicability of the these models in the practical setting.

2. Elaboration of a case study showing a systematic development of predictors for system faulty modules from failure history of other projects from the field, and successful use of product complexity metrics to predict these failures. This includes the experiences and lessons learned from systematic empirical investigation of available data, which will provide guidance in several software engineering decisions and further strengthen the existing empirical body of knowledge in software engineering.

## 1.6 Scope and Limitations

The scope of this dissertation is confined to the investigation of an adaptive fault prediction approach and supporting design methodology for identifying faulty modules belonging to systems for which historical fault data are not available. We neither promote specific software quality measure, nor do we suggest that the underlying statistical methods used in this dissertation for fault prediction produce best results. Rather, the usability of the presented fault prediction approach in commercial setting is the primary focus of the work. The use of software quality measurements and fault detection techniques are provided as means to validate presented approach.

Furthermore, findings identified in this dissertation hold across the investigated open-source domain projects. It may not be possible to extend the findings of the study involving open-source software systems to proprietary software due to the different development practices adopted [40]. Further validations with both open-source and proprietary software systems are necessary to help us draw stronger conclusions.

## 1.7 Outline of the Dissertation

The rest of this dissertation is organized as follows: Chapter 2 provides the background and related work on the problem under investigation. Chapter 3 presents the data collection process in detail. We present the development of the predictive models in Chapter 4. Our adaptive fault prediction approach is explained in Chapter 5. The results of our study are presented in Chapter 6. Chapter 7 concludes the research investigation and discusses future work.

# CHAPTER 2

# LITERATURE REVIEW

In this chapter we provide background material (Section 2.1) essential to understanding the problem under investigation and summarize the prior literature in the area of software measurement and prediction (Section 2.2).

## 2.1 Background

Our research introduces an approach that, for a given project, selects and *adapts* the appropriate training dataset used in the development of a fault-proneness prediction model, improving its ability to predict fault-prone modules across projects. As predictive models rely on the observation of historical data obtained through measurements, in the next section we relate the process of measurement to software systems. Statistical analysis used for development of predictive models is described in Section 2.1.2.

### 2.1.1 Software Measurement

Fenton et al. [24] define the measurement as the process by which numbers are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules. He further introduces a notion of software metrics as a collective term used to describe the very wide range of activities concerned with measurement in software engineering.

Software engineering incorporates three distinct entities: (1) *processes* or collection of software-related development activities, (2) *products* or artifacts, deliverables and documents resulting from processes, and (3) *resources* required by a process activity. Each entity has a set of related attributes. Software metrics produce numbers

that characterize an attribute of a software engineering entity, and can therefore be classified into three categories:

1. *Process metrics* inform on duration, cost, effectiveness and efficiency of the software development process. Examples include the duration of the process or activity and the effort associated with process or activity.

2. *Product metrics* quantify some attribute of artifacts, deliverables and documents resulting from software development process in terms of size, complexity, and design features.

3. *Resource metrics* describe the project resources like personnel, materials and methods required by software development process. Examples include programmer's productivity and skill level.

Additionally, within each entity we distinguish between *internal* attributes, measurable purely in terms of the entity itself, separate from its behavior, and *external* attributes, measurable only with respect to how the entity relates to its environment [24]. Internal product attributes measure product in terms of size, length, and functionality for example and are generally easily obtainable prior to actual deployment of the system, but of little use unless related to product external attributes. External attributes are concerned with product quality, such as usability, testability, reusability, and portability [24]. As external attributes are directly observable only after the system has already been deployed and operational for some time, the focus has been on relating internal attributes (these are the classic software metrics) to their external qualities.

To support their usefulness in practical applications, it is necessary to empirically validate software metrics [23]. Empirical studies frequently rely on data sets. However, large software data repositories from which representative samples can be

drawn are often accessible only internally in a company or organization doing software development. Even though open source software projects are exception, their development methodology, often performed by volunteers, is different from the usual formalized methods applied by all team members in commercial software development [30]. Hence their process and resource data are often incomplete and unreliable. Consequently, to conduct and validate our empirical work, we exploit product metrics only.

### 2.1.2 Statistical Methodology

Regression analysis is a statistical tool for the investigation of relationships between dependent and independent variables (also called predictors). Regression models are widely used for prediction. In regression model, the dependent variable is assumed to be a function of one or more independent variables, called the regression function:

$$Y \approx f(X, \beta) \tag{2.1}$$

where $X$ represents a set of independent (or explanatory) variables, $Y$ is the dependent (or response) variable, and $\beta$ denotes unknown parameters. Given a data set of known $X$ and $Y$ values, regression analysis estimates the values of unknown parameters $\beta$ to fit a regression model.

Logistic regression approaches have been widely used to estimate the impact of various independent variables (OO metrics) on the dependent variable (faults) in prior studies [6, 21, 22, 58].

**Logistic regression:** When the dependent variable $y_i$ within our data set assumes only two values (yes/no), logistic regression analysis provides a model to predict the probability $p_i$ for a specific event for $y_i$ (fault-prone) given the values of

$k$-vector of regressors $x_i$ (OO metrics). The multiple logistic regression function has the following form:

$$\ln(\frac{p_i}{1-p_i}) = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + ... + \beta_k x_{ik} \tag{2.2}$$

Parameters of a logistic regression model are usually estimated using the maximum likelihood method. For a fixed set of data and underlying probability model, maximum likelihood picks the values of the model parameters that make the data "more likely" than any other values of the parameters would make them. Within the scope of this work, we use the logistic regression model.

## 2.2 Related Work

First, we introduce several metrics suites for OO software that were relevant to fault prediction in past studies. In Section 2.2.2 we overview and compare prior literature on fault prediction using OO metrics.

### 2.2.1 Object-Oriented Metrics

Development under the OO paradigm has spurred a burst in the availability of OO product metrics. Generally, each metric is considered as either a *size* or a *design* measure. Size metrics typically measure some attribute of software code while design metrics relate to various OO design constructs. In this section we introduce two most commonly cited OO class metrics suites in the literature.

Abreu et al. [1] presented a suite of metrics for OO design (MOOD) that do not depend to a great extent on the definitions of functions, so they can be collected early in the design phase. The metrics values are independent of the system size, and hence return values between 0 (absence of a factor) and 1 (the maximum possible presence of a factor). Selected MOOD class metrics are described in Table 2.1.

Table 2.1: Selected OO Class Metrics Definitions

| Abreu's MOOD Metrics [1] | |
|---|---|
| Metric Name | Definition |
| Attribute Hiding Factor (AHF) | The ratio of (1 - total number of accessible attributes) to the total number of attributes in a class. |
| Method Hiding Factor (MHF) | The ratio of (1 - total number of accessible methods) to the total number of methods in a class. |
| Attribute Inheritance Factor (AIF) | The ratio of inherited attributes to the total number of attributes in a class. |
| Method Inheritance Factor (MIF) | The ratio of inherited methods to the total number of methods in a class. |
| **Chidamber and Kemerer's (CK) Metrics [17]** | |
| Metric Name | Definition |
| Weighted Methods per Class (WMC) | The sum of the weights of all methods in a class. If all method weights are unity, same as number of methods. |
| Depth of Inheritance Tree (DIT) | The maximum distance in the inheritance tree of a given class from the root node of hierarchy. |
| Number of Children (NOC) | The number of children classes inheriting directly from a given class. |
| Coupling Between Objects (CBO) | Counts other classes whose members are used by a given class + those that use the members of a given class. |
| Response for a Class (RFC) | Counts all local methods of a class + all methods on other classes directly called by any methods on a given class. |
| Lack of Cohesion of Methods (LCOM) | Number of disjoint sets of local methods where any two methods on same set share at least one local variable. |

The metrics defined by Chidamber and Kemerer [17] (CK) cover many aspects of the OO paradigm and are most widely validated in fault prediction studies. Table 2.1 presents selected size and complexity CK metrics representing characteristics of the OO code.

### 2.2.2 Fault Prediction Techniques

The general principle behind prediction model development is depicted in Figure 2.1. Projects with available code and fault histories allow us to map their faults back to

individual modules. The metrics for each module involved in the mapping process are then computed. Relationships between metrics and faults are analyzed. The result is a prediction model, which can then be used to estimate the fault probability of new modules.

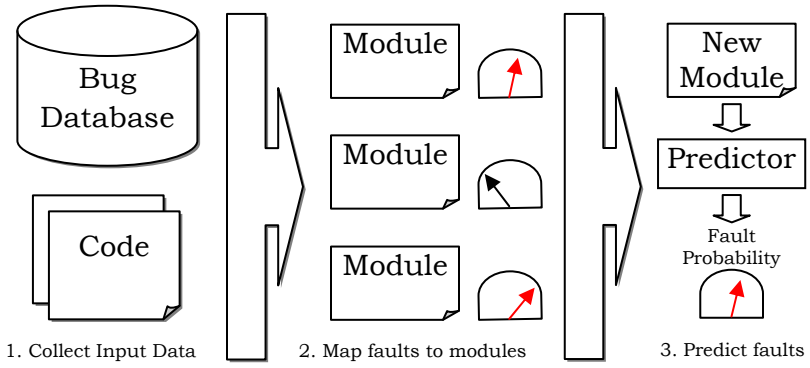


Figure 2.1: Prediction Uses Historical Faults to Predict Faults of New Modules

Numerous studies have empirically validated the association between OO metrics and faults. The selected literature included in this section is most related to our work and includes OO metrics based prediction models that focus on validating the effectiveness of OO metrics for either predicting fault-prone classes or number of faults in the classes. We present related studies in chronological order, starting with earliest and ending with most recent literature.

Basili et al. [6] conducted experiments on eight medium-sized student C++ projects for which they collected fault data during acceptance testing. The metrics under the investigation were the CK metrics and logistic regression was utilized to perform the data analysis. By dividing classes into two distinct categories, faulty (contained one or more faults) and non-faulty (fault free), they showed that CK metrics were statistically independent and all CK metrics except LCOM were significantly associated with class fault-proneness.

Tang et al. [58] conducted their study on data from an industrial system comprised of over 200 subsystems implemented in C++. In addition to CK metrics, additional metrics were investigated, and logistic regression was carried out to evaluate those metrics. The results illustrated that WMC and RFC were significant indicators of fault-prone classes, but CK metrics alone were not sufficient predictors of quality.

Emam et al. [21] also chose logistic regression to analyze data from a telecommunication system consisting of 174 C++ classes. They found WMC, RFC, and CBO were closely associated with fault-proneness. However, they also demonstrated that the significance of the metrics no longer existed when size controlling was imposed on the analysis, urging other researchers to reexamine their studies.

Yu et al. [63] completed another validation study of CK metrics with data from the client side application of a large network service management system, which contained 123 Java classes and approximate 34,000 lines of code. The dependent variable was the number of faults present in a class, and linear regression (ordinary least square) and linear discriminant analysis were their analysis methodology. They came to conclusion that except DIT all CK metrics were sound predictors of fault-prone classes.

Subramanyam et al. [56] validated the association between WMC, CBO, and DIT metrics and the fault counts, rather than fault-proneness. They analyzed around 400 C++ and 300 Java classes, concluding that CK metrics were significantly associated with faults counts, but they found that effectiveness of these metrics vary in the two programming languages investigated. While C++ classes found WMC, DIT, and interaction term (CBO*DIT) all significantly associated with faults counts, Java classes were significantly associated with faults through interaction term (CBO*DIT) only.

Gyimothy et al. [30] also validated the CK metrics as significantly associated with class fault-proneness using logistic and linear regression, and machine learning. They investigated an open source software Mozilla version 1.6 by inspecting over 3,000 C++ classes and investigating their fault histories collected from the Bugzilla database. All metrics except NOC were significant predictors of class fault-proneness in their study.

Zhou et al. [65] explored C++ data set from the public NASA data set repository and took severity of faults into account when researching the relationship between CK metrics and fault-prone classes. Their analysis method was logistic regression and machine learning. They maintained that all CK metrics except DIT were significant regardless of severity levels. Moreover, they held that severity levels could greatly influence the predicting power of CK metrics upon class fault-proneness.

Olague et al. [46] empirically validated three sets of metric suites to predict fault-proneness of OO classes using highly iterative or agile software development process: CK metrics, Abreu's Metrics for OO Design (MOOD) [2], and Bansiya and Davis' quality metrics for OO design (QMOOD) [5]. They used fault data for six versions of Rhino, an open source implementation of JavaScript written in Java, and concluded that the CK and QMOOD metric suites both produce logistic regression models that are effective in detecting fault-prone classes. Their study showed WMC and RFC as consistent predictors of fault-prone classes across all versions of Rhino, while CBO was significant for five, LCOM for four, and both DIT and NOC for two out of six versions. In their study, MOOD metric suite was not effective in detecting fault-prone classes.

Xu et al. [62] utilize linear regression and a neuro-fuzzy approach to validate relationships between CK metrics suite and number of faults in OO classes. Investigated applications belong to the public NASA data set repository and are implemented in

C++. Their results indicate that all CK metrics but DIT are reliable metrics for estimating the number of faults present in a class. Overall, they report that SLOC metric imposes most significant impact on the number of faults.

English et al. [22] explored fault-proneness of a Java-based open-source subsystem belonging to an integrated development environment Eclipse, using logistic regression. Again, the CK metrics were selected as quality predictors. They conclude that inheritance based metrics NOC and DIT were not very useful in any of the prediction models. The coupling metrics CBO and RFC were the best predictors of fault prone classes in their study.

A summary of the selected literature is listed in Table 2.2. We can easily see that (1) the fault-proneness is the most employed dependent variable choice among researchers, (2) logistic regression is most frequently utilized statistical method, and (3) WMC, CBO, and RFC are most widely accepted as useful indicators for faults or fault-prone classes.

Table 2.2: Summary of Related Literature

| Study | Method | Dependent Var. | WMC | DIT | NOC | CBO | RFC | LCOM |
|---|---|---|---|---|---|---|---|---|
| Basili et al. [6] | LR | Fault-proneness | + | + | + | + | + | - |
| Tang et al. [58] | LR | Fault-proneness | + | - | - | - | + | - |
| Emam et al. [21] | LR | Fault-proneness | + | - | - | + | + | - |
| Yu et al. [63] | OLS+LDA | Faults | + | - | + | + | + | + |
| Subramanyam et al. [56] | OLS | Faults | + | + | x | + | x | x |
| Gyimothy et al. [30] | LR+ML | Fault-proneness | + | + | - | + | + | + |
| Zhou et al. [65] | LR+ML | Fault-proneness | + | - | + | + | + | + |
| Olague et al. [46] | OLS+LR | Fault-proneness | + | - | - | + | + | + |
| Xu et al. [62] | OLS | Faults | + | - | + | + | + | + |
| English et al. [22] | LR | Fault-proneness | + | - | - | + | + | + |

Method legend: LR-Logistic regression; ML-Machine Learning; LDA-Linear Discriminant Analysis
OLS-Ordinary Least Square. Metrics legend: + (predictor); - (not predictor); x (not used)

Even though there is a general consensus over the limited use of existing models to predict faults across different software projects, to our knowledge, effort to combine or modify existing models in order to improve their cross-project performance has virtually been nonexistent within published literature. The most notable work is by

Bouktif et al. [10]. They use a genetic algorithm based approach to improve the cross-project performance of existing models using a combinations of several existing models. Their results show that combining existing models can yield significantly better results than using any of the existing models individually.

# CHAPTER 3

# MAKING A CORPUS: DATA COLLECTION

The key component behind this dissertation is learning from software evolution history. Fault prediction models in this dissertation are based on software history mining that involves the extraction of useful information from software evolution data. Most software projects today use some type of software configuration management (SCM) system to manage and record the evolution data. For the open source projects we analyzed, the SCM system includes either Concurrent Versions (CVS) [9] or Subversion (SVN) [8] component.

Within both CVS and SVN components, each developer has their own isolated working space called a workspace. The workspace is a directory on the developers' local workstation. From the SCM systems, developers check out source code into their own workspace. Developers usually change files in their workspace and when they want to store their changes in the SCM system, they submit their changes using the commit command. Changes made in workspaces are not visible to other developers until changes are committed to the SCM system, and then each developer updates their local workspace from the SCM system to apply changes from others. When developers commit changes, they can commit more than one file change at the same time. A group of changes at the same time is called a revision. The details of each revision are recorded in the shared change log, and include author, date, a list of changed files, and log message indicating the type of revision. For change log example, please refer to Figure 3.1.

Since each project's evolution data also includes years of fault fix revisions, they can be a good resource for predicting faults, by learning from previous mistakes. We refer to a collection of analyzed systems and their respective data as a corpus. Corpus

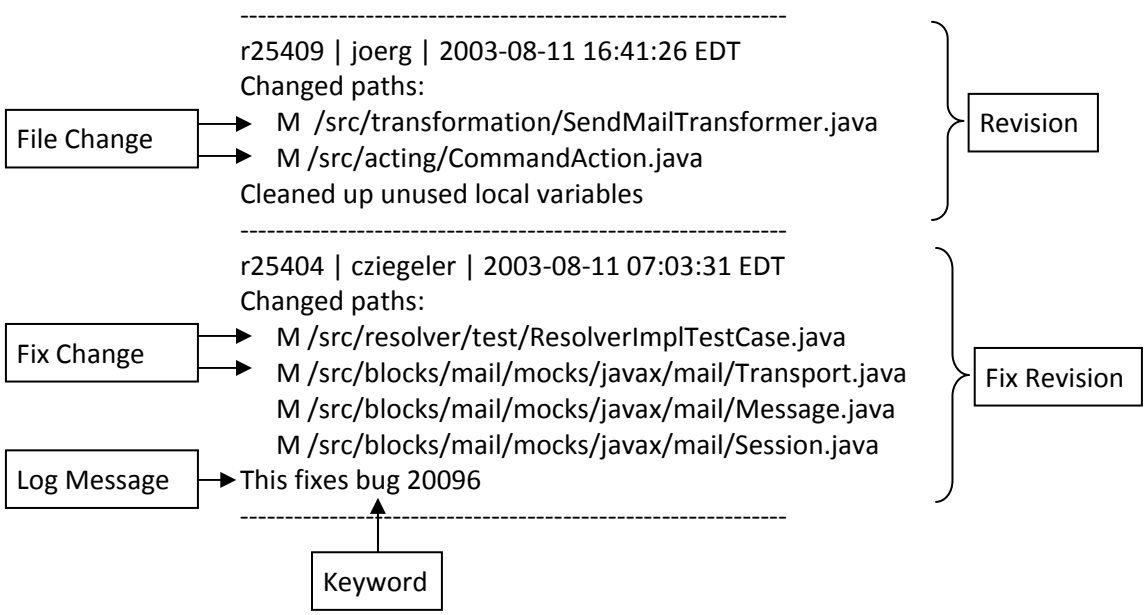


Figure 3.1: Change Log Entries Example

is described in detail in this chapter, and for each system within it includes project's complete source code (Section 3.1), its fault fix revisions (Section 3.2), and relevant OO metrics (Section 3.3).

## 3.1 Target Applications

Open source projects listed in Table 3.1 are used in this dissertation, and are all developed in Java. These projects are chosen due to availability of their full source code and the entire project evolution history.

A more detailed description of each project is described as follows:

- Cacoon (http://cocoon.apache.org/) - Apache Cocoon is a web application framework built around the concepts of pipeline, separation of concerns and component-based web development. Full source code and change code entries are available through SVN at http://svn.apache.org/repos/asf.

Table 3.1: Open Source Projects Analyzed

| Project | Version | Software Type | SCM | Class Count |
|---|---|---|---|---|
| Cacoon | 2.1 | Web development framework | SVN | 1961 |
| Columba | 1.0 | Email client | SVN | 1941 |
| Cosmos | 1.0 | System management framework | CVS | 1828 |
| Derby | 10.3 | Relational database | SVN | 2488 |
| Eclipse JDT | 3.1 | Java development tools | CVS | 3410 |
| JEdit | 4.0 | Text editor | SVN | 642 |
| OpenOffice | 3.1 | Office suite | SVN | 537 |
| Scarab | 1.0 | Bug tracker | SVN | 475 |

- Columba (http://sourceforge.net/projects/columba/) Columba is an email client, featuring a user-friendly graphical interface with wizards and internationalization support. Its a powerful mail management tool. Full source code and change code entries are available through SVN at https://columba.svn.sourceforge.net/svnroot/columba/.

- Cosmos (http://www.eclipse.org/cosmos/) - The Cosmos (Community-driven Systems Management in Open Source) project aims to provide an extensible, standards-based framework upon which software developers can create specialized, differentiated and inter-operable offerings of tools for system management. Full source code and change code entries are available through CVS using :pserver:anonymous@dev.eclipse.org:/cvsroot/technology.

- Derby (http://db.apache.org/derby/) - Apache Derby is an open source relational database. Full source code and change code entries are available through SVN at http://svn.apache.org/repos/asf.

- Eclipse JDT (http://www.eclipse.org/jdt/) Eclipse is a universal and extendable integrated development environment (IDE) for software development. The JDT project is the part of Eclipse and provides the tool plug-ins that implement

a Java IDE, supporting the development of any Java application. Full source code and change code entries are available through CVS using :pserver:anonymous@dev.eclipse.org:/cvsroot/eclipse.

- JEdit (http://www.jedit.org/) JEdit is a programmers text editor that supports plug-ins. It is a highly configurable and customizable editor. Full source code and change code entries are available through SVN at https://jedit.svn.sourceforge.net/svnroot/jedit.

- OpenOffice (http://www.openoffice.org/) - OpenOffice is an open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. Full source code and change code entries are available through SVN at svn://svn.services.openoffice.org/ooo.

- Scarab (http://scarab.tigris.org/) Scarab is a Bugzilla-like bug tracking system that is highly customizable. Full source code and change code entries are available through SVN at http://scarab.tigris.org/svn/scarab/trunk.

## 3.2 Fault History Extraction

A fault is created during the development process and causes abnormal software behavior. These abnormal behaviors are often reported by users and developers, and are typically recorded in a tracking system such as Bugzilla [47]. Developers then locate the fault and fix it by changing one or more files related to the fault. Traditionally, faults are identified in software by examining the output from software execution, performing software inspections, or running static analysis tools. Developers are assumed to have been using these traditional methods for fault identification throughout a project's evolution, and have been fixing the faulty code. The method

for fault identification used in this dissertation thus relies on identifying these revisions, which constitute fault fixes. We will refer to such revisions as fix revisions.

There are some heuristic ways to identify fix revisions in software evolution history. They rely on the history logs (Figure 3.1) left by the developers [26, 54, 60]. If the change logs can provide any clue that indicates that the revision is fixing some problem, it is then considered a fix revision. We use a simple algorithm that parses project's change log for special keywords that indicate fixes, such as *Fixed* or *Fault* [43], and for references to fault reports like *#1234567* [26, 54, 60]. This heuristic identifies whether an entire revision contains a fault fix. If it does, all files in the revision are marked as fix changes. Manual inspection of the change logs for each project is used to identify the keywords that indicate fix revisions for each project. The project keywords are shown in Table 3.2.

Table 3.2: Keywords and Reference Identifiers Used to Find Fix Revisions

| Project | Keywords or Phrases |
|---|---|
| Cacoon | patch, fix, bug |
| Columba | [bug], [bugfix] |
| Cosmos | bug |
| Derby | patch, fix, bug |
| Eclipse JDT | * bug id reference |
| JEdit | patch, fix, bug |
| OpenOffice | * bug id reference |
| Scarab | patch, fix, bug, issue number |
| * Bug id reference is a 7-digit number. | |

As developers record which files have been changed within the history log, rather than which specific classes have been modified, in this study we exclude revisions to files containing two or more class definitions, and exclude their classes from the scope of this investigation. There is no way to determine whether a revision is attributed to the public class with the same name as its encompassing file, or some other non-public class definition within the same file. Thus, attributing some specific complexity

metric value of the public class contained within the same-named file to the fix revision performed on a completely different class would negatively influence the validity of this study.

## 3.3 Generating Object-Oriented Metrics

As discussed in Section 2.2.1, most widely validated metrics in fault prediction studies across different projects are metrics defined by Chidamber and Kemerer [17]. While our future work might include investigating other software quality measures and their correlation to faults, in this dissertation we investigate five CK metrics that have shown to be most suitable for predicting fault prone OO classes. They are WMC, CBO, RFC, DIT, and LCOM. Again, their full definitions are discussed in detail in Table 2.1 of Section 2.2.1.

We calculate metric values by using a third party software tool Understand [51] by Scientific Toolworks. It includes a set of static analysis tools that measure various metric calculations, including CK metrics. We then generate descriptive statistics summary for a given project in terms of its metrics using IBM SPSS [35] software for predictive analysis, which is also used for regression analysis and development of prediction models.

# CHAPTER 4

# RAW DATA PREDICTION MODELS

In this study we first describe the process involved in developing raw data fault prediction models using CK metrics. We chose to use logistic regression [34] instead of the traditional linear regression technique to discover the relationships between the values of the metrics and the fault-proneness of classes. Thus, we are predicting fault-proneness as a dichotomous response variable. As previously noted in Section 2.1.2, the logistic regression method only predicts if a class is faulty or not, but does not say anything about the possible number of faults in that class. We develop logistic regression models through SPSS statistical software tool [35] using the forward stepwise regression method. Again, as stated in Section 1.6, we do not make the case in favor of the regression method used. Rather, the usability of the presented fault prediction approach in commercial setting is the primary focus of the work. Logistic regression does not assume linearity of relationship between the independent (CK metrics) and dependent (class fault-proneness) variable nor does it require normal distribution assumption. Logistic regression has been shown to provide good models for fault-proneness prediction in previous studies [20].

## 4.1 Data Distribution

In order to determine the appropriate statistical procedures for our data analysis and the overall research conducted in this dissertation, it is imperative that we establish whether the values of our metrics belong to a normally distributed population, also

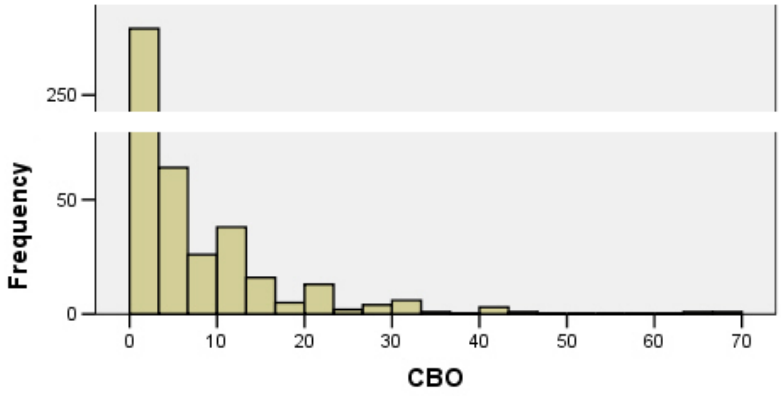


Figure 4.1: Histogram for Scarab CBO

known as Gaussian. Manipulating nonparametric[1] data requires a different set of statistical tools than manipulating its parametric[2] counterpart.

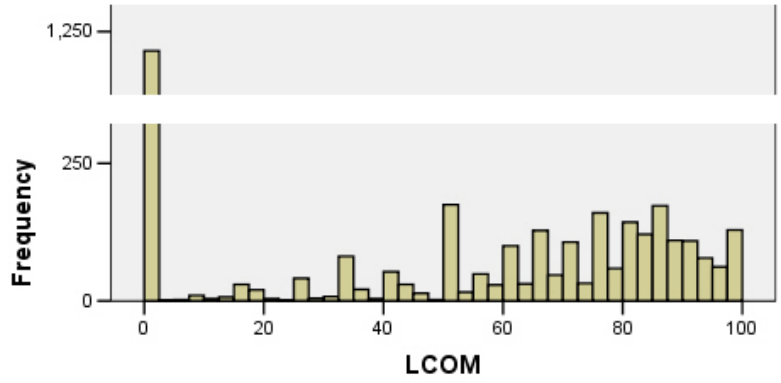


Figure 4.2: Histogram for Eclipse LCOM

To obtain a visual impression of the distribution of metrics data, we generated their graphical representation in the form of histogram. Histogram is an approximation of the probability distribution of a variable and consists of tabular frequencies, shown as adjacent rectangles, erected over discrete intervals, with an area equal to

[1]We do not make any assumptions about the population distribution of our metrics data. Metrics data do not belong to any specific distribution such as Gaussian.

[2]Normal distribution assumption

the frequency of the observations in the interval [33]. We present an example of histogram generated for the distribution of three metrics, CBO metric within the project Scarab (Figure 4.1), then for the distribution of LCOM metric within the project Eclipse (Figure 4.2), and finally for the distribution of WMC metric within the project Columba (Figure 4.3) respectively. Please note that the chunk of the histogram Frequency scale is missing, without any information being omitted. Otherwise, the figures would have been unnecessarily high. From Figures 4.1, 4.2, and 4.3 we observe that our data are skewed. We therefore rely on nonparametric statistical approach.

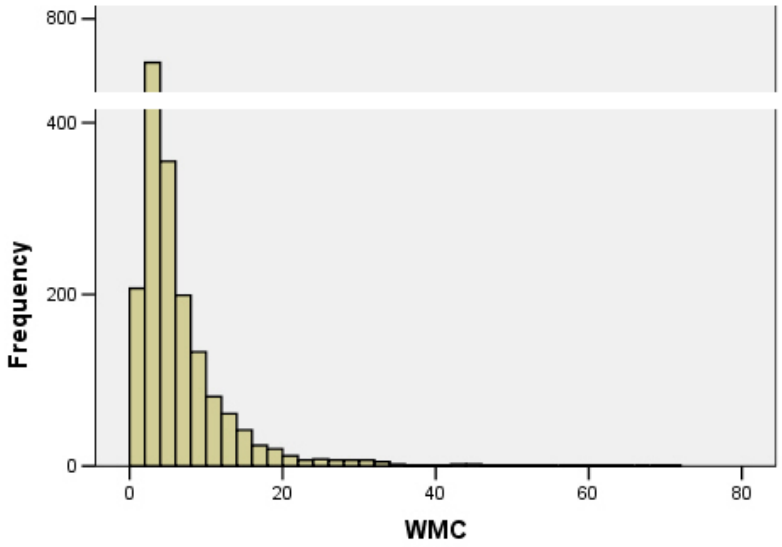


Figure 4.3: Histogram for Columba WMC

## 4.2 Comparison of CK Metrics Interrelationships

In order to determine if individual CK metrics measure different quality properties of software, we compared inter-correlation between the individual metrics within the CK metric suite using the Spearman Rank's correlation. Spearman Rank's correlation is a nonparametric statistical measure of dependence between two variables

[18]. The complete results of the Spearman Rank correlation are presented in the Appendix, Table A1 for projects Cocoon, Columba, Cosmos, and Derby, and Table A2 for projects Eclipse, JEdit, OpenOffice, and Scarab. Table 4.1 summarizes the correlation for all six CK metrics across all eight projects by evaluating the correlation coefficients against the Hopkins criteria [61](<0.10 trivial (T), 0.11-0.30 minor (S), 0.31-0.50 moderate (M), 0.51-0.70 large (L), 0.71-0.90 very large (VL), 0.91-1.00 almost perfect (P)).

Our correlation results indicate that across all projects, RFC at times showed very large correlation with WMC and DIT metrics, and large correlation with CBO metric. NOC metric shows trivial to minor correlation with the remaining metrics within the suite. CBO metric shows medium to large correlation with WMC and RFC, while it remains unrelated to remaining metrics. No two metrics show perfect correlation. Results suggest that WMC and RFC metrics are the only that consistently show significant correlation when compared to one another across all projects.

Table 4.1: Bivariate Spearman Interrelationships between CK Metrics

| CBO | NOC | WMC | RFC | DIT | LCOM | Metric |
|---|---|---|---|---|---|---|
| 1 | T-S | S-L | M-L | S-M | S-M | CBO |
| | 1 | T-S | T-M | T-S | T-S | NOC |
| | | 1 | L-VL | T-S | M-L | WMC |
| | | | 1 | M-VL | M | RFC |
| | | | | 1 | T-S | DIT |
| | | | | | 1 | LCOM |
| Legend: | T=Trivial(<0.10); S=Minor(0.11-0.30); M=Moderate(0.31-0.50); L=Large(0.51-0.70); VL=Very Large(0.71-0.90); P=Perfect(0.91-1.00); | | | | | |

## 4.3 Individual Metrics as Fault Predictors

In this subsection, we performed binary logistic regression (LR) analysis of the metrics versus faults on all projects within our corpus to identify which CK metrics were statistically significant fault-proneness indicators. In turn, each individual CK metric

was used as the independent variable in the regression. The dependent variable was a boolean variable representing whether or not classes were classified as faulty. Following guidelines for most suitable logistic regression sample size selection as suggested by Long [39], we ensure that each training sample extracted from individual projects and used in regression contains the same number of faulty and non-faulty classes.

Table 4.2: Class Sample Sizes Used in Regression

| Project | Total Class Count | Classified as Faulty | Sample Size |
|---|---|---|---|
| Cacoon | 1961 | 15% | 500 |
| Columba | 1941 | 12% | 450 |
| Cosmos | 1828 | 34% | 600 |
| Derby | 2488 | 26% | 600 |
| Eclipse | 3410 | 36% | 1000 |
| JEdit | 642 | 30% | 200 |
| OpenOffice | 537 | 32% | 200 |
| Scarab | 475 | 38% | 300 |

Table 4.2 indicates the size of the class training dataset used for each project, as well as the total number of classes within the project and what percent of the total number of classes within the project were classified as faulty. We randomly selected the equal number of classes from the faulty and non-faulty subsets of the project classes to arrive at the training dataset used in the regression. For each faulty and non-faulty subset of classes, we created a *Collection* structure, *ArrayList*, and loaded it with the appropriate classes. We then used the *shuffle* static method within *Collections* class that resides in *java.util* package within the JDK 5.0 [57]. Following a randomized shuffle, we retrieved the first $n$ elements from each collection, arriving at the sample size for each project equaling $2n$ classes, as indicated within the Table 4.2 under column titled Sample Size.

The complete results of the LR analysis for CK metrics are shown in the Appendix, Table A3 for projects Cocoon, Columba, Cosmos, and Derby, and Table A4

Table 4.3: Logistic Regression Results Summary

| Project | CBO | NOC | WMC | RFC | DIT | LCOM |
|---|---|---|---|---|---|---|
| Cocoon | S | N | SG | S | S | S |
| Columba | SG | N | SG | N | S | SG |
| Cosmos | SG | N | S | S | S | S |
| Derby | S | N | S | S | N | S |
| Eclipse | S | N | S | S | S | S |
| JEdit | SG | N | S | S | S | SG |
| OpenOffice | SG | S | S | N | N | SG |
| Scarab | SG | N | SG | S | SG | S |

Legend: N=Not Significant(p-value≥0.05); S=Significant(p-value<0.05); SG=Significant(p-value<0.05) and Good Fit(HL p-value≥0.05);

for projects Eclipse, JEdit, OpenOffice, and Scarab. The results are summarized in Table 4.3. Consistent with the results presented in Section 2.2, only three investigated metrics (CBO, WMC, and LCOM) have coefficients that are significant at the $\alpha$=.05 level for all projects we examined. NOC metric was not significant for all but OpenOffice project. RFC and DIT metrics were significant for some projects, while not significant for other projects. We used Hosmer-Lemeshow (HL) test as an inferential goodness-of-fit test [34], which is also contained in the Appendix Tables A3 and A4 with its chi-square values, degrees of freedom (DF) and $p$-values. It showed a good fit of data ($p$-value>0.05) for CBO (Columba, Cosomos, JEdit, OpenOffice, and Scarab), WMC (Cacoon, Columba, and Scarab), DIT (Scarab), and LCOM (Columba, JEdit, and OpenOffice). We were quite surprised, that otherwise metric identified as good predictor in previous studies [6, 22, 62, 46], RFC has not exhibited even a single good fit of data value produced by the HL test as an inferential goodness-of-fit test. In the wake of our findings, we select three metrics as best candidates for our logistic regression and ultimately the development of our fault prediction models: CBO, WMC, and LCOM.

## 4.4 Combined Metrics as Fault Predictors

Olague et al. [46] emphasized that multiple logistic regression, which considers several metrics together within a single prediction model, may produce better fault-proneness prediction than a binary logistic regression model developed from a single metric alone. In order to evaluate our adaptive approach to fault-prediction against their approach, we also develop multiple regression models and compare their predictive ability with our results.

In order to identify which metrics to use in our multiple logistic regression models, we performed a collinearity analysis to determine if there are any potential collinearity problems in the bivariate correlations between the CK metrics within their CK metric suite. When there is a perfect linear relationship among the predictors, the estimates for a regression model cannot be uniquely computed. The term collinearity implies that two variables are near perfect linear combinations of one another. The primary concern is that as the degree of multicollinearity increases, the regression model estimates of the coefficients become unstable and the standard errors for the coefficients can get wildly inflated. We compute the variance inflation factor (VIF) values for each CK predictor as a check for multicollinearity, which is the reciprocal of the tolerance. The tolerance is an indication of the percent of variance in the predictor that cannot be accounted for by the other predictors. A commonly given rule of thumb is that VIF value of 10 or higher may be reason for multicollinearity concern. This is, however, just a rule of thumb; Allison [3] says he gets concerned when the VIF is over 2.5. The VIF values within this work, as suggested by University of Kentucky Information Technology [36], if greater than 3.0 may merit further investigation of potential regressors for multicollinearity problems. We present the VIF analysis for the CK metrics suite in Table 4.4.

Table 4.4: Collinearity Analysis for CK Metrics

| Metric | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
|---|---|---|---|---|---|---|---|---|
| VIF Values (Using all CK metrics) | | | | | | | | |
| CBO | 1.1868 | 1.4442 | 1.2839 | 1.5373 | 1.8568 | 1.7564 | 1.1399 | 2.2140 |
| NOC | 1.0041 | 1.0107 | 1.0256 | 1.0068 | 1.0338 | 1.0343 | 1.0650 | 1.0147 |
| WMC | 2.5228 | 1.5221 | 2.6055 | 1.8869 | 2.2542 | 21.0076 | 2.8213 | 7.2215 |
| RFC | 3.0025 | 2.8242 | 3.7338 | 2.6052 | 1.9160 | 20.2597 | 3.4745 | 6.4637 |
| DIT | 1.9840 | 2.9294 | 2.2318 | 1.9279 | 1.3817 | 1.4461 | 2.0522 | 2.0545 |
| LCOM | 1.2916 | 1.3568 | 1.2122 | 1.2381 | 1.1628 | 1.3305 | 1.3438 | 1.4131 |
| VIF Values (Removing Metric RFC) | | | | | | | | |
| CBO | 1.1792 | 1.4441 | 1.2792 | 1.4770 | 1.4770 | 1.7379 | 1.1341 | 2.2039 |
| NOC | 1.0030 | 1.0099 | 1.0156 | 1.0068 | 1.0068 | 1.0334 | 1.0470 | 1.0147 |
| WMC | 1.3332 | 1.5089 | 1.3700 | 1.4929 | 1.4929 | 1.6418 | 1.2629 | 2.4238 |
| DIT | 1.0799 | 1.1179 | 1.0764 | 1.0351 | 1.0351 | 1.1007 | 1.0990 | 1.1348 |
| LCOM | 1.2914 | 1.3567 | 1.2009 | 1.2351 | 1.2351 | 1.3206 | 1.3390 | 1.4008 |

The table shows VIF analysis with all potential regressors. Collinearity problems are suspected with WMC and RFC since they exceed our VIF threshold for all projects except Columba, Derby, and Eclipse. As RFC includes the count of all local methods of a class, which are also accounted for in WMC, the VIF results are expected. We thus need to remove one of the regressors from the model. The univariate binary logistic regression analysis presented in the Section 4.3 shows both WMC and RFC as significant indicators of quality for all projects. However, given the results of the Hosmer-Lemeshow (HL) test (inferential goodness-of-fit test) also presented in Section 4.3, we selected WMC as the better predictor candidate. RFC did not show a good fit of data for even a single project within our project set. After removing RFC from the model, a subsequent VIF analysis shows (Table 4.4) that the remaining variables are within VIF threshold and objective values. The results in Section 4.3 showed NOC metric as not significant across most projects, and DIT metric showing a good fit of data for Derby project solely. Based on these results, we develop and explore the performance of one multiple LR model for CK metrics which includes CBO, WMC, and LCOM as its predictors.

Table 4.5: Multiple LR Results Summary

| Project | CBO | WMC | LCOM | HL |
|---|---|---|---|---|
| Cocoon | S | N | S | G |
| Columba | S | N | N | G |
| Cosmos | S | S | N | - |
| Derby | S | S | N | - |
| Eclipse | S | N | S | - |
| JEdit | S | N | S | G |
| OpenOffice | S | N | S | G |
| Scarab | S | S | N | G |

Legend: N=Not Significant(p-value≥0.05);
S=Significant(p-value<0.05);
HL=Hosmer-Lemeshow goodness of fit test;
G=Good data fit (HL p-value≥0.05);

To determine coefficients of selected CK metrics for our combined metrics fault prediction models, we performed LR on CBO, WMC, and LCOM versus faults for each project analyzed in this dissertation. Detailed results of the multiple LR analysis are presented in the Appendix, Table A5. The results are summarized in Table 4.5. None of the projects had all investigated CK metrics as significant regressors ($p$-value<0.05). CBO and LCOM were significant regressors in the multiple LR model for projects Cocoon, Eclipse, JEdit and OpenOffice. CBO and WMC were significant regressors in the multiple LR model for Cosmos, Derby, and Scarab. And project Columba had a single significant regressor CBO.

## 4.5 Fault Prediction Models

Given the coefficient values presented in the Appendix Tables A3 and A4, we developed twenty-four binary logistic regression models to predict fault-proneness of classes. Their coefficients' values along with their constants are presented in Table 4.6.

Table 4.6: Binary Logistic Regression Coefficients

| Models | CBO | | WMC | | LCOM | |
|---|---|---|---|---|---|---|
| | Const. | Coeff. | Const. | Coeff. | Const. | Coeff. |
| Cocoon | -0.603 | 0.193 | -0.530 | 0.075 | -0.653 | 0.016 |
| Columba | -1.208 | 0.096 | -0.384 | 0.058 | -0.507 | 0.014 |
| Cosmos | -0.495 | 0.139 | -0.825 | 0.129 | -0.402 | 0.010 |
| Derby | -0.921 | 0.092 | -0.871 | 0.064 | -0.738 | 0.015 |
| Eclipse | -0.687 | 0.081 | -0.585 | 0.055 | -0.711 | 0.015 |
| JEdit | -0.931 | 0.255 | -0.689 | 0.119 | -1.084 | 0.031 |
| OpenOffice | -0.683 | 0.134 | -0.395 | 0.050 | -0.531 | 0.014 |
| Scarab | -1.651 | 0.345 | -1.393 | 0.204 | -0.677 | 0.020 |

Given the presented values of the constants and the coefficients for individual metrics, we show the example of the general CBO fault prediction model developed using data for project Cocoon as:

$$\ln(\frac{p}{1-p}) = -0.603 + 0.193 * CBO \tag{4.1}$$

where the ln symbol refers to a natural logarithm.

Therefore, we calculate the probability $p$ that the class is faulty as follows:

$$p = \frac{e^{(-0.603+0.193*CBO)}}{1+e^{(-0.603+0.193*CBO)}} \tag{4.2}$$

Similarly, given the coefficient values presented in the Appendix Table A5, we developed eight multiple logistic regression models to predict fault-proneness of classes. Their coefficients' values along with their constants are presented in Table 4.7.

Given the presented values of the constants and the coefficients for individual metrics, we show the example of the general CBO fault prediction model developed using data for project Cocoon as:

$$\ln(\frac{p}{1-p}) = -0.990 + 0.165 * CBO + 0.011 * LCOM \tag{4.3}$$

Table 4.7: Multiple Logistic Regression Coefficients

| Models | Const. | CBO Coeff. | WMC Coeff. | LCOM Coeff. |
|---|---|---|---|---|
| Cocoon | -0.990 | 0.165 | - | 0.011 |
| Columba | -1.208 | 0.096 | - | - |
| Cosmos | -1.021 | 0.101 | 0.104 | - |
| Derby | -1.181 | 0.062 | 0.041 | - |
| Eclipse | -1.008 | 0.066 | - | 0.009 |
| JEdit | -1.537 | 0.187 | - | 0.025 |
| OpenOffice | -0.955 | 0.119 | - | 0.009 |
| Scarab | -1.990 | 0.281 | 0.100 | - |

Therefore, we calculate the probability $p$ that the fault in a class is present as follows:

$$p = \frac{e^{(-0.990+0.165*CBO+0.011*LCOM)}}{1+e^{(-0.990+0.165*CBO+0.011*LCOM)}} \tag{4.4}$$

We present the evaluation of the models in the following section.

## 4.6 Models Evaluation

We evaluate the performance of the models with a threshold value of 0.5, which means that, if probability $p \geq 0.5$, the class is identified as fault-prone. Otherwise, if probability $p < 0.5$, the class is identified as not fault-prone [11]. We evaluate models across three standard measures used for classification techniques: *recall*, *precision* and *F-score* [42].

To understand the three measures essential to evaluating model performance, we define the following variables:

- True Positives (TP): The number of faulty classes correctly identified as faulty.
- True Negatives (TN): The number of non-faulty classes correctly identified as non-faulty.

- False Positives (FP): The number of non-faulty classes incorrectly identified as faulty.
- False Negatives (FN): The number of faulty classes incorrectly identified as non-faulty.

The *recall* $R$ (also called *sensitivity*) is defined as:

$$R = \frac{TP}{TP + FN} \tag{4.5}$$

Recall measures the proportion of the actual faulty classes which are correctly identified as such. From a theoretical point of view, a prediction model which always indicates positive result of a class being fault-prone, regardless of the actual fault-proneness status of the class, will achieve 100% *recall*. However, such result says nothing about how many non-faulty classes were identified as faulty in the process. Therefore the *recall* alone cannot be used to determine whether a prediction model is useful in practice [42].

Consequently, we define the *precision* $P$ as:

$$P = \frac{TP}{TP + FP} \tag{4.6}$$

Precision measures the proportion of the classes identified as faulty that are actually faulty. In contrast to *recall*, a model achieving a perfect *precision* score of 100% indicates that all classes identified as faulty are indeed faulty, but says nothing about whether all faulty classes were identified in the process [42]. Therefore, *recall* and *precision* scores are not discussed in isolation. Instead, both are combined into a single measure, such as their harmonic mean, the *F-score*.

The *F-score*, the harmonic mean of the *recall* $R$ and the *precision* $P$, is defined as:

$$F = 2 * \frac{R * P}{R + P} \tag{4.7}$$

As a single measure of model's performance, the *F-score* can be interpreted as a weighted average of the precision and recall, reaching its best value at 1 and worst value at 0.

Table 4.8: Model Evaluation across all Projects

| **Cocoon** | Recall | Precision | F-Score | **Columba** | Recall | Precision | F-Score |
|---|---|---|---|---|---|---|---|
| CBO | 46% | 26% | 0.33 | CBO | 62% | 26% | 0.37 |
| WMC | 48% | 30% | 0.37 | WMC | 41% | 18% | 0.25 |
| LCOM | 64% | 22% | 0.32 | LCOM | 60% | 17% | 0.26 |
| MRM | 58% | 24% | 0.34 | MRM | 62% | 26% | 0.37 |
| **Cosmos** | Recall | Precision | F-Score | **Derby** | Recall | Precision | F-Score |
| CBO | 47% | 74% | 0.58 | CBO | 57% | 46% | 0.51 |
| WMC | 55% | 76% | 0.64 | WMC | 54% | 53% | 0.53 |
| LCOM | 59% | 62% | 0.61 | LCOM | 67% | 34% | 0.45 |
| MRM | 57% | 75% | 0.65 | MRM | 60% | 51% | 0.55 |
| **Eclipse** | Recall | Precision | F-Score | **JEdit** | Recall | Precision | F-Score |
| CBO | 52% | 60% | 0.56 | CBO | 56% | 46% | 0.50 |
| WMC | 48% | 53% | 0.50 | WMC | 53% | 60% | 0.56 |
| LCOM | 68% | 45% | 0.54 | LCOM | 71% | 53% | 0.61 |
| MRM | 60% | 55% | 0.58 | MRM | 72% | 54% | 0.62 |
| **OpenOffice** | Recall | Precision | F-Score | **Scarab** | Recall | Precision | F-Score |
| CBO | 43% | 91% | 0.58 | CBO | 67% | 87% | 0.76 |
| WMC | 38% | 55% | 0.45 | WMC | 53% | 74% | 0.62 |
| LCOM | 63% | 64% | 0.63 | LCOM | 55% | 64% | 0.59 |
| MRM | 52% | 75% | 0.61 | MRM | 64% | 85% | 0.73 |
| Legend: | MRM = Multiple regression model; | | | | | | |

In this section, we first estimate the predictive power of our models by cross validating [55] against the entire dataset from which the subset training dataset was drawn during the model construction. In other words, both training dataset used in the construction of the prediction model and the testing dataset which the prediction model is evaluated against belong to the same project. Table 4.8 shows the values for the *recall*, *precision* and *F-Score* measures of the logistic regression fault prediction models (CBO, WMC, and LCOM as individual predictors, and multiple LR model which combines all three metrics) when applied to the entire project testing dataset.

However, to obtain realistic estimates of our model applicability in practical setting, we must employ *cross-system validation*, which consists of applying models to testing datasets other than those the models were derived from [11]. Hence, in the following section, we use *F-scores* as our threshold values to apply and assess the prediction models derived from one project's training dataset (e.g. Cocoon) to a testing dataset belonging to an entirely different project (e.g. Eclipse).

# CHAPTER 5

# ADAPTIVE APPROACH TO FAULT PREDICTION

In this chapter, we first apply the prediction models that are built in the previous chapter to test datasets belonging to entirely different projects (Section 5.1). Hence, using *cross-system validation* [11], we evaluate the practical applicability of the existing prediction techniques across entirely different projects. In Section 5.2 we present the descriptive statistics of the metrics as design measures and their analysis. The process of selecting an appropriate training dataset from our repository by identifying similarly distributed datasets is described in Section 5.3. We then preform metrics data transformations and the development of calibrated models in Section 5.4, and then re-evaluate obtained models on our transformed testing datasets in Section 5.5.

Table 5.1: Cross-System Validation Results Summary for CBO

| Model: CBO | Testing Dataset | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | - | - | Y | Y | Y | Y | Y | Y |
| Columba | - | - | - | - | - | - | - | - |
| Cosmos | Y | - | - | Y | Y | Y | Y | Y |
| Derby | - | - | - | - | - | - | - | - |
| Eclipse | - | - | - | Y | - | - | - | - |
| JEdit | Y | - | Y | Y | Y | - | Y | Y |
| OpenOffice | - | - | - | Y | Y | - | - | - |
| Scarab | Y | - | - | Y | Y | - | Y | - |
| Legend: Y = model is applicable; | | | | | | | | |

## 5.1 Cross-System Model Validation

In this section, we investigate whether prediction models built from the history and metrics of one project are applicable to other unrelated projects. Hence, every raw-data model developed in Section 4.5 has been in turn applied to the set of classes of each of the other seven remaining projects. Using F-Score baseline values presented in Section 4.6 and obtained when both training and testing datasets belonged to

the same project, we calculate and compare F-Score values of using the same models across all projects. We then analyze our results to see which of the raw data models can be considered reasonably appropriate model for unrelated project under investigation.

Table 5.2: Cross-System Validation Results Summary for WMC

| Model: WMC | Testing Dataset | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | - | Y | - | - | Y | - | Y | - |
| Columba | - | - | Y | - | Y | - | Y | Y |
| Cosmos | - | Y | - | - | Y | - | Y | Y |
| Derby | - | - | - | - | - | - | - | - |
| Eclipse | - | - | - | Y | - | - | - | - |
| JEdit | - | - | Y | - | Y | - | Y | Y |
| OpenOffice | Y | - | - | - | Y | - | - | - |
| Scarab | - | Y | Y | - | Y | - | Y | - |

Legend: Y = model is applicable;

The complete results of F-Scores values obtained during the cross-system validation process are presented in the Appendix, Table A6. Consistent with findings reported by Nagappan et al. [44], some project histories can serve as predictors for other projects, while most cannot. This is true for prediction models derived from the metrics CBO, WMC, and particularly for multiple LR model. Our findings for CBO, WMC, and multiple LR model prediction across projects is summarized in Table 5.1, Table 5.2, and Table 5.3 respectively. The $Y$ entry indicates that the model derived from one project is applicable (the F-Score value is either equal or higher than its baseline value counterpart) to the indicated unrelated project. The frequency of $Y$ entries is comparable between CBO, WMC, and multiple LR models. We noticed that the increased complexity of the model makes it less likely to be applicable across different projects, as demonstrated by the least number of $Y$ entries within the multiple LR model in Table 5.3. From a general standpoint, this suggests that, in the common situations where development practices and project domains are evolving, an absolute and general interpretation of predicted probabilities is not possible when they come from prediction models built from different systems.

As such, we are prevented from using a predetermined cut-off value and predicted probabilities to classify classes according to their fault-proneness. And as one would intuitively suspect, learning from earlier fault history data can only be successful if the two projects are similar, and sharing the same heterogeneous metric and fault distribution across comprising modules.

Table 5.3: Cross-System Validation Results Summary for Multiple LR Models

| Model: Multi. | | | | Testing Dataset | | | | |
|---|---|---|---|---|---|---|---|---|
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | - | - | Y | - | Y | - | Y | Y |
| Columba | - | - | - | - | - | - | - | - |
| Cosmos | Y | - | - | - | Y | - | Y | Y |
| Derby | - | - | - | - | - | - | - | - |
| Eclipse | Y | - | - | - | - | - | - | - |
| JEdit | - | - | Y | - | Y | - | Y | Y |
| OpenOffice | Y | - | - | - | Y | - | - | - |
| Scarab | Y | - | - | - | Y | - | - | - |

Legend: Y = model is applicable; Multi. = Multiple LR model

However, the LCOM model showed some rather interesting results, as indicated within its cross-validation summary within Table 5.4. We have observed a substantially larger number of $Y$ entries across almost all projects. In order to understand the discrepancy in findings, in the following section, we present and investigate descriptive statistic for metrics and fault distributions, and devise strategy for identifying similar projects.

Table 5.4: Cross-System Validation Results Summary for LCOM

| Model: LCOM | | | | Testing Dataset | | | | |
|---|---|---|---|---|---|---|---|---|
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | - | Y | Y | Y | Y | - | Y | - |
| Columba | Y | - | Y | Y | Y | Y | Y | Y |
| Cosmos | Y | Y | - | Y | Y | - | Y | Y |
| Derby | Y | Y | Y | - | Y | - | Y | - |
| Eclipse | Y | Y | Y | Y | - | - | Y | - |
| Jedit | Y | Y | Y | Y | Y | - | Y | Y |
| OpenOffice | Y | Y | Y | Y | Y | Y | - | - |
| Scarab | Y | Y | Y | Y | Y | Y | Y | - |

Legend: Y = model is applicable;

## 5.2 Descriptive Statistics and Interpretation

The object oriented metrics measure unique aspects of the object oriented approach and the complexity of the design. To explain why some project histories can serve as predictors for some projects, but not others, we analyzed the individual metric measures within our target applications using box plots. A box plot, also known as a box-and-whisker diagram, is a suitable way of graphically depicting groups of numerical data through their five-number summary statistics: the smallest observation (sample minimum), lower quartile[1] (representing $25^{th}$ percentile), median, upper quartile (representing $75^{th}$ percentile), and the largest observation (sample maximum). Box plots display differences between populations without making any postulations of the underlying statistical distribution [33]. The spacing between the different parts of the box assist in identifying the degree of dispersion[2] and skewness[3] in the data.

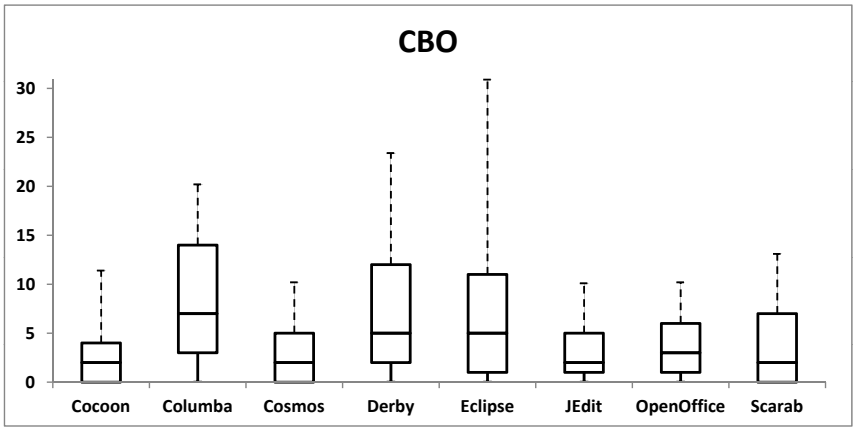


Figure 5.1: CBO Box Plot

The box plots for each investigated metric used in the development of prediction models across all projects are presented in Figure 5.1 for CBO metric, Figure 5.2 for

[1] In descriptive statistics, the quartiles of a set of values are the three points that divide the data set into four equal groups, each representing a fourth of the population being sampled

[2] Dispersion measures variability or spread of metric data distribution [33]

[3] Skewness measures asymmetry of the metric data distribution [33]

WMC metric, and Figure 5.3 for LCOM metric respectively. Notice that diagram whiskers extending from the upper quartile to the largest observation are represented with a dotted line for both CBO and WMC box plots. Due to the consequent clarity of the figure presented, we decided to depict whiskers in a proportionally reduced size. As a result, indicated whiskers, while suitably proportional to one another, do not indicate their factual value within the graph. For exact values of box plot statistics, please refer to Table A7 in the Appendix.

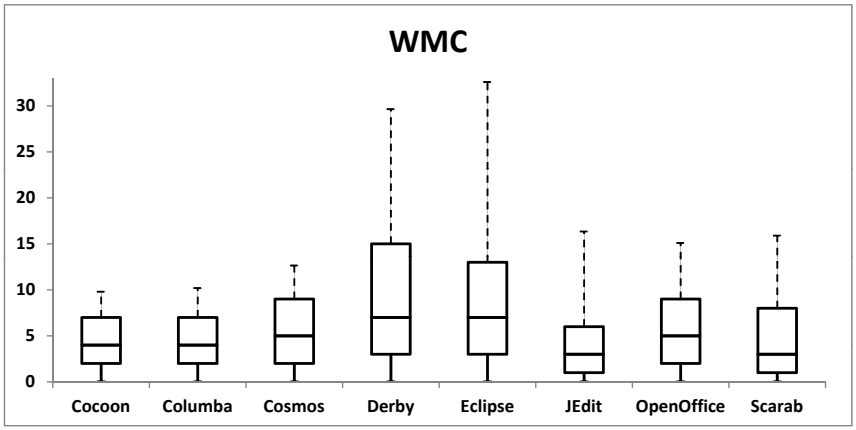


Figure 5.2: WMC Box Plot

The box plots suggest that there is a tendency for data spread variability across different projects. Moreover, we have observed that as the number of classes comprising the project increases, so does the probability of its data being more dispersed. This was in particular true for CBO and WMC metrics, as evident in Figures 5.1 and 5.2. However, we noted that LCOM metric did not experience a great degree of variability across projects, regardless of the individual size of the target applications. The lack of variability of LCOM data from one project to another may intuitively explain why did the LCOM prediction models derived from one project prove consistently applicable to other unrelated projects, as demonstrated by the high number of $Y$ entries within the Table 5.4.

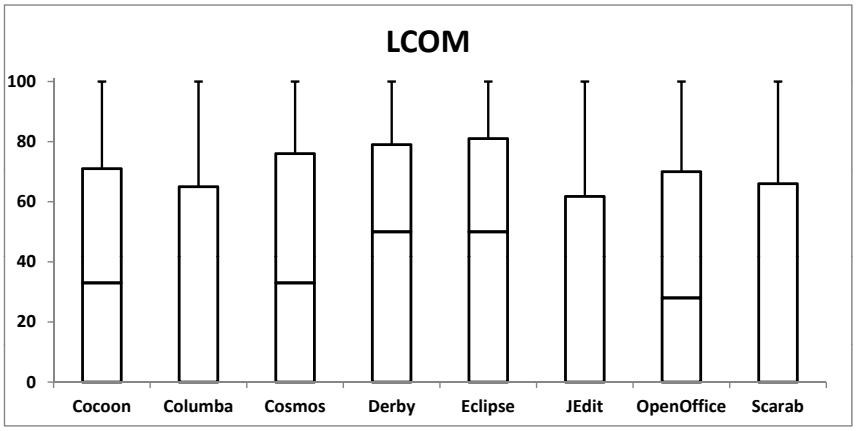


Figure 5.3: LCOM Box Plot

Following the analysis of the box plots, in particular considering CBO and WMC metrics, we conclude that the ability of prediction models for fault-proneness derived from the context-specific OO metrics is diminished when applied to other, unrelated projects. To demonstrate the problem at hand, let suppose that we have an existing LR model which considers the WMC as predictor of fault-proneness. And let's assume that it has been derived from a training dataset belonging to a large sized project $A$, where the faulty classes have been identified as those with WMC$\geq$12. Our attempt to use the same prediction model on a much smaller project B, with a maximum WMC value of 11, would produce an empty dataset for classes identified fault-prone, as illustrated in Figure 5.4. We attempt to solve this problem in the following two section, by first identifying projects that share similarly distributed datasets, and then by employing simple log transformations further making the value of metrics comparable among unrelated projects.

## 5.3 Model Selection: Identifying Similarly Distributed Datasets

The first step in addressing the problem statement illustrated in Figure 5.4 involves identifying projects that share the same heterogeneous metric distribution across com-

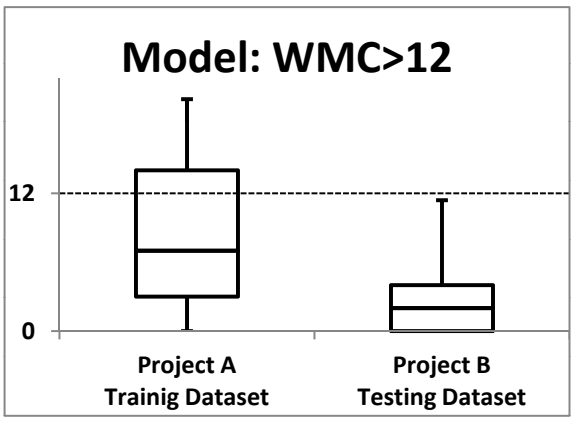


Figure 5.4: Context-Specific Fault Prediction

prising modules. Assume we are given some project $P$ for which the prior fault history data is not known and we are interested in computing fault-proneness probabilities across its modules. Additionally, assume we have a set of projects $P_i'$ with known fault histories that can serve as training datasets for the development of classifiers. How do we select the most appropriate single project from the $P_i'$ for $P$, as to avoid problem statement illustrated in Figure 5.4? We have observed that the most appropriate project selection is such that the single selected project from the $P_i'$ set and project $P$ share similar measure of how far a set of its particular metric values is spread out from the median value. Therefore, we are interested in testing the equality of variances between two data samples.

In this dissertation, we use the nonparametric Levene's Test based on rank transformations [19] as an inferential statistic to assess the equality of variances in different metric samples. In general terms, given a variable $Y$ with sample of size $N$ divided into $k$ subgroups, where $N_i$ is the sample size of the $i^{th}$ subgroup, the Levene's Test statistic $F$ is defined as:

$$F = \frac{(N-k)}{(k-1)} \frac{\sum_{i=1}^{k} N_i (\overline{Z}_{i.} - \overline{Z}_{..})^2}{\sum_{i=1}^{k} \sum_{j=1}^{N_i} (Z_{ij} - \overline{Z}_{i.})^2} \tag{5.1}$$

where $\overline{Z}_{i\cdot}$ are the group means of the $Z_{ij}$, $\overline{Z}_{\cdot\cdot}$ is the overall mean of the $Z_{ij}$, $Z_{ij} = |Y_ij - \widetilde{Y}_{i\cdot}|$, and $\widetilde{Y}_{i\cdot}$ is the median of the $i^{th}$ subgroup. We use median values instead of traditional mean values, as median values showed to perform better when the underlying data followed skewed distribution, while mean provided better results for symmetric, slightly-tailed distributions [13, 52].

If the resulting p-value of Levene's test is less than level of significance 0.05, the obtained differences in sample variances are unlikely to have occurred based on random sampling. Thus, the null hypothesis of equal variances is rejected and it is concluded that there is a difference between the variances in the population.

We have calculated Levene's Test statistics on every single project pair across three metrics CBO, WMC, and LCOM. The complete results of the test with their $F$ statistics and *p-values* is presented in Table A8 in the Appendix. Most of the *p-values* within the table are less than 0.05, indicating that substantial differences between metric variances of the two projects exist. However, for each project, except fully for Cosmos, we were able to find a corresponding project that showed p-values larger than 0.05, indicating high probability of data's equality of variances. The summary of our findings is presented in Table 5.5. For example, given the Cocoon project as the project for which we are interested computing fault-proneness probabilities, and a set of remaining seven projects as possible training datasets for the development of a classifier appropriate for Cacoon, we determined that the variance of the Cocoon's CBO metric is most comparable to the CBO metric of OpenOffice. Therefore, we would use the fault histories and CBO metric values belonging to OpenOffice for the development of the CBO classifier appropriate to generate Cocoon's fault-proneness probabilities. However, please note that CBO variance equality did not imply WMC variance equality. Rather than being similar to OpenOffice across all design-complexity measures and consistent with our findings in Section 5.1, Cocoon's

WMC metric was most comparable to the WMC metric of JEdit. Therefore, we would use the fault histories and WMC metric values belonging to JEdit for the development of the WMC classifier appropriate to generate Cocoon's fault-proneness probabilities. The only project for which we have not identified a statistically significant comparable project was Cosmos for metrics CBO and LCOM. However, we can still assign the preferred project, indicated in the parenthesis, as the one exhibiting the highest statistically non-significant p-value.

Table 5.5: Levene's Test Results Summary

| Project | Equality of Variances | | |
|---|---|---|---|
| | CBO | WMC | LCOM |
| Cocoon | OpenOffice | JEdit | Columba |
| Columba | Scarab | OpenOffice | Scarab |
| Cosmos | (Derby) | Cocoon | (Eclipse) |
| Derby | Scarab | Scarab | Eclipse |
| Eclipse | Scarab | Scarab | Derby |
| JEdit | OpenOffice | Cocoon | Scarab |
| OpenOffice | Cocoon | Scarab | Scarab |
| Scarab | Eclipse | Derby | JEdit |

## 5.4 Model Adaptation: Metrics Data Transformations

The second step in addressing the problem statement illustrated in Figure 5.4 involves further reducing variability and promoting equality of spread among the datasets through the use of statistical data transformation techniques. Statistical data transformation includes the application of a deterministic mathematical function to each point in a data set – that is, each data point $m_i$ is substituted with the transformed value $m_i' = f(m_i)$, where $f$ is some transformation function [33].

In the previous studies [16], power transformations have showed to be a preferable choice used to stabilize variance, make the data more normal distribution-like, and

improve the validity of measures. Generally, power transformation function transforms every metric data point $m$ by either $m^p$ for $p$ values greater than zero, or by $\log(m)$ for $p$ values equal to zero [33]. In this dissertation, we chose to use the power transformation function with the value of *p=0*. However, given that many of our metrics assume zero values, simply using the $\log(m)$ transformation would produce infinite results. We solve this by a simple and commonly used transformation, $m' = \log(m + 1)$ [33]. In addition, we align testing dataset with its appropriate training dataset, so that both have a common reference point – their median value. Thus, the final transformation function we use on our datasets is $m' = \log(m+1)+d$, where $m$ is the initial metric value to be transformed, $d$ is the difference between the median values of the transformed datasets, and *m'* is the new transformed value. The box plot example of the transformation applied to the CBO metric for two projects identified as having CBO similarly distributed Cocoon and OpenOffice is presented in Figure 5.5.

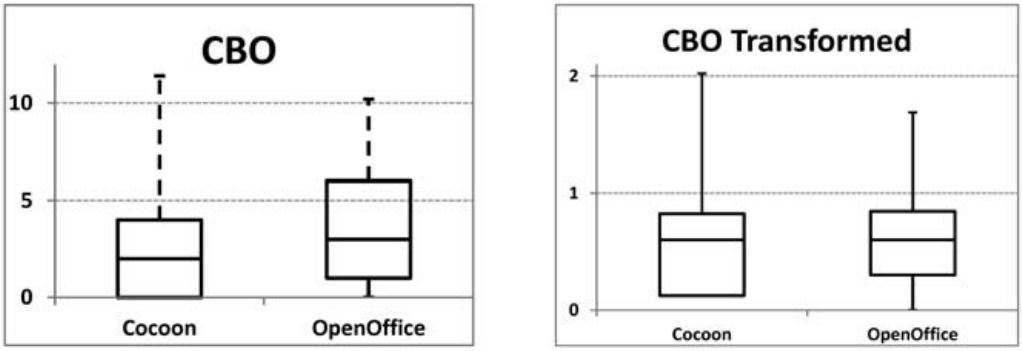


Figure 5.5: CBO Box Plot Before and After the Transformation

For the project pairs identified in previous section, we performed logistic regression analysis identically as in Section 4.3, but this time using project's transformed datasets. The detailed results of the logistic regression analysis are presented in Table A9 in the Appendix, containing the regressors' coefficients, their corresponding

p-values, as well as chi-square values and p-values of the inferential goodness-of-fit HL test. In the next section, we perform their evaluation and compare their performance with results previously obtained in Section 4.6.

## 5.5 Evaluation of Models with Transformed Datasets

First, we wanted to assess the overall goodness of fit for our new models developed using transformed datasets. We use the Hosmer-Lemeshow (HL) test as an inferential goodness-of-fit test [34], as we did during logistic regression performed on our raw data in Section 4.3. Similarly, several datasets showed a very good fit of data ($p$-value>0.05) for CBO (Cocoon, Derby, Eclipse), WMC (Cocoon, Derby, JEdit, OpenOffice, Scarab), and LCOM (Columba, Derby, JEdit). In order to determine whether our transformed dataset models achieve better fit of data, we compared their HL statistics and p-values (presented in Table A9 in the Appendix) against their respective counterparts (presented in Tabels A3 and A4 in the Apendix) obtained during the logistic regression analysis performed on raw data. Results are mixed. For investigated CBO metric, Cocoon, Derby, and Eclipse projects exhibited good fit of data for their transformed datasets, but not their raw datasets. However, during the transformation process, the CBO good fit of data test values decayed for projects OpenOffice and Scarab, which showed very good fit of data for their raw values. In addition to transforming the fit of data from no good to good for Cocoon and Derby, metric WMC was the only one to show consistent improvements in goodness of fit test across the board for all investigated models. However, except for one single project Derby, LCOM transformed data showed consistent decay in goodness of fit test values across all prediction models that are generated.

Following the goodness of fit analysis, we compared the predictive ability of our new transformed classifiers against their raw data counterparts. The summary of the

Table 5.6: The Evaluation Summary of Transformed Data Models

| Project: | | Cocoon | | Project: | | Columba | |
|---|---|---|---|---|---|---|---|
| Model from: | OpenOffice | JEdit | Columba | Model from: | Scarab | OpenOffice | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Effect | + | = | - | Effect | + | = | = |
| F1-Score | 0.32 | 0.36 | 0.32 | F1-Score | 0.28 | 0.24 | 0.25 |
| F2-Score | 0.35 | 0.36 | - | F2-Score | 0.36 | 0.24 | 0.25 |
| Project: | | Cosmos | | Project: | | Derby | |
| Model from: | Derby | Cocoon | Eclipse | Model from: | Scarab | Scarab | Eclipse |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Effect | + | + | + | Effect | = | + | - |
| F1-Score | 0.34 | 0.59 | 0.61 | F1-Score | 0.51 | 0.51 | 0.46 |
| F2-Score | 0.63 | 0.67 | 0.62 | F2-Score | 0.51 | 0.53 | 0.44 |
| Project: | | Eclipse | | Project: | | JEdit | |
| Model from: | Scarab | Scarab | Derby | Model from: | OpenOffice | Cocoon | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Effect | = | - | + | Effect | + | + | = |
| F1-Score | 0.57 | 0.57 | 0.45 | F1-Score | 0.49 | 0.51 | 0.61 |
| F2-Score | 0.57 | 0.50 | 0.55 | F2-Score | 0.51 | 0.62 | 0.61 |
| Project: | | OpenOffice | | Project: | | Scarab | |
| Model from: | Coccoon | Scarab | Scarab | Model from: | Eclipse | Derby | JEdit |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Effect | = | - | - | Effect | + | + | = |
| F1-Score | 0.66 | 0.51 | 0.63 | F1-Score | 0.65 | 0.43 | 0.59 |
| F2-Score | 0.66 | 0.45 | - | F2-Score | 0.79 | 0.67 | 0.59 |
| Legend: + Model Improvement; - Model Decay; = No Change | | | | | | | |

observations is presented in Table 5.6. The CBO classifiers showed an improvement in predicting fault-prone classes across five different projects using transformed datasets: Cocoon (using OpenOffice's classifier), Columba (using Scarab's classifier), Cosmos (using Derby's classifier), JEdit (using OpenOffice's classifier), and Scarab (using Eclipse's classifier). Their improvement is denoted with a +(plus) entry within the Table 5.6. For Derby (using Scarab's classifier), Eclipse (using Scarab's classifier), and OpenOffice (using Cocoon's classifier), the transformed classifiers achieved identical predictive power, as measured by their F-Scores and indicated by the =(equal) entry within the Table 5.6. WMC classifiers showed an improvement over four projects: Cosmos (using Cocoon's classifier), Derby (using Scarab's classifier), JEdit (using Cocoon's classifier), and Scarab (using Derby's classifier). Except for the Eclipse (using Scarab's classifier) project, where transformed classifier experienced decay (denoted by -(minus) entry within the Table 5.6), the WMC classifiers for transformed datasets showed no change for the remaining three projects. Our final investigated metric

LCOM showed improvement across only two projects: Cosmos (using the Eclipse's classifier) and Eclipse (using Derby's classifier). While remaining unchanged across three projects, Columba (using Scarab's classifier), JEdit (using Scarab's classifier), and Scarab (using JEdit's classifier), consistent with its values exhibited by the goodness of fit test, LCOM showed decay for three remaining projects. Two of those however, showed a complete collapse of the transformed classifier.

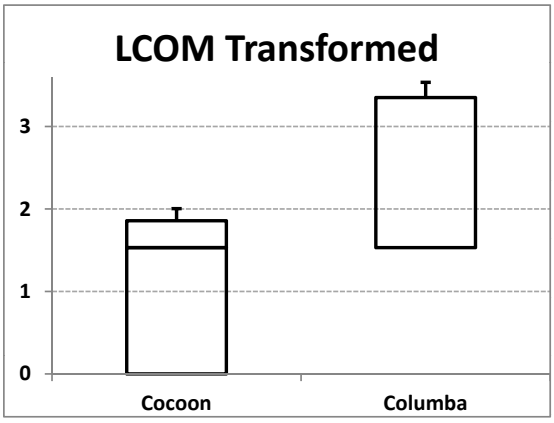


Figure 5.6: LCOM Box Plot After the Transformation

To make sense of these observations, we first wanted to address LCOM metric, as it is clearly the one that has exhibited behavior that is contrary to what we were trying to achieve in the first place. We point out that LCOM metric is unlike the other two metrics investigated in this dissertation, as demonstrated by its histogram in Figure 4.2 and its box plot in Figure 5.3. The frequency of LCOM data values are in great deal concentrated at the value of zero or the other end of the interval in which the values appear (0-100), across all projects from our application dataset. Our transformation technique minimizes variable variance between the two datasets by transforming them to a single common reference point: the higher median value of the two. Since median values between datasets might vary significantly, as they will always amount to either a value of zero or value at the higher end of the interval

LCOM assumes, at times data transformation produces an exact opposite of what we intend to do. We demonstrate that in Figure 5.6 by the comparison of LCOM transformed datasets between Cosmos and its matched project Cocoon, from which we developed appropriate LCOM classifier. Such scenario is certain to bring about an entire collapse of the classifier's ability to predict any fault-prone modules within the system, as presented in Table 5.6 for projects Cocoon (using Columba's classifier) and OpenOffice (using Scarab's classifier). We thus conclude that the data transformation technique used in this dissertation is not an appropriate choice for LCOM metric. Perhaps, using power transformation with mean values instead median values as common reference points might generate better results, but given the results presented in Table 5.4 and LCOM's consistent ability to achieve reasonable classification results across different projects, such investigation is outside of the scope of this study.

In general, log transformations for metrics CBO and LCOM improved the prediction results, as their measures were not as spread as those used in the construction of the raw dataset in Chapter 4. Out of eight transformed CBO models evaluated, five showed an improvement in identifying fault-prone classes, while the remaining three models stayed unchanged. We observed no deterioration of transformed CBO classifiers. WMC metric did not perform as well. Out of eight transformed WMC models evaluated, four showed an improvement in identifying fault-prone classes, while two remained unchanged, and two showed deteriorated performance. And considering the complete collapse of transformed classifiers across two projects, LCOM metric showed the least degree of improvement, achieving higher predictive results for only two out of eight models evaluated. The two remained LCOM transformed classifiers remained unchanged. In the following chapter, we discuss the results of our study.

# CHAPTER 6

# RESULTS

In this chapter, we discuss the findings (Section 6.1) in terms of lessons learned and in terms of the extent to which the research goal has been accomplished, and the implications (Section 6.2) in terms of practical suggestions for addressing the issues that have been raised in the research. Threats to validity are presented in Section 6.3. Framework for fault prediction is discussed in Section 6.4.

## 6.1 Findings

One of the main objectives of this dissertation was to propose an approach assisting developers to use fault-proneness models, based on design measurements, as viable decision making tools when applied from one object-oriented system to the other. During our investigation, we applied a fault-proneness classifiers developed using the training dataset from one project to another unrelated project.

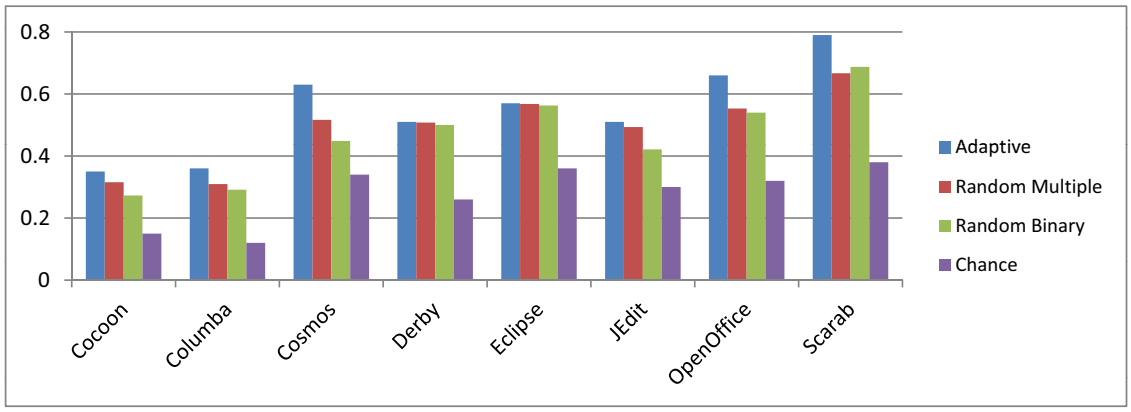


Figure 6.1: CBO Model F-Scores for Different Fault Classifier Choices

Our results suggest that choosing OO metrics without a proper validation is unlikely to predict fault-prone modules. On the other side, OO metrics did prove to be

useful as abstractions over program source code, capturing similarity over OO classes that turned out to be reasonable indicators of module's fault-proneness. Even though the projects used in this research stem from the same open-source development environment, the distributions of OO measures change significantly from one project to other, making the task of predicting across projects difficult to achieve. As a solution, we develop an approach consisting of a systematic selection technique complemented by the data transformation technique to construct adapted classifiers applicable across unrelated projects. Recall that our criterion for the research objective identified in Section 1.3 requires that the predictive accuracy, as measured by the F-Score values, of the fault-proneness prediction model developed using our approach is greater than the accuracy of a randomly selected raw-data fault-proneness prediction model or a model predicting fault-prone class modules by chance.

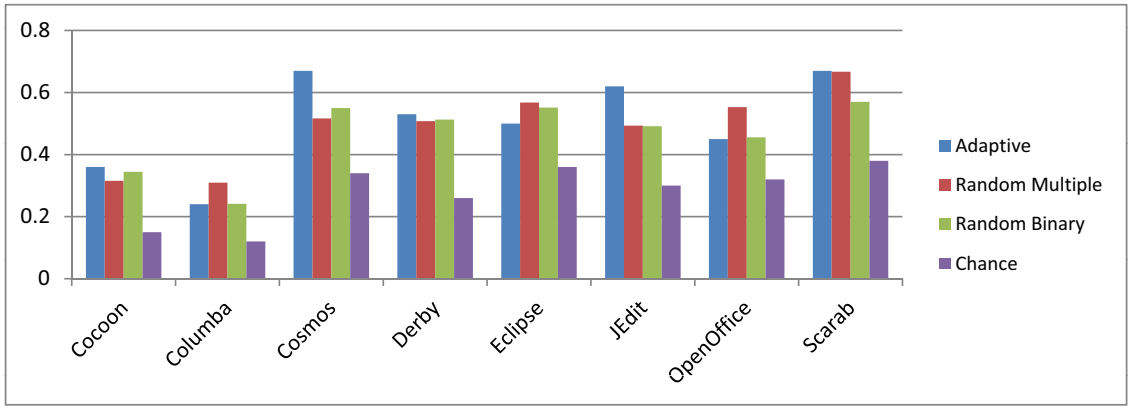


Figure 6.2: WMC Model F-Scores for Different Fault Classifier Choices

Figure 6.1 illustrates side-by-side comparison of the performance of our fault-proneness prediction models based on CBO metric and expressed in models' F-Score values. The series within figure labeled as *Adaptive* represents the performance of the appropriate classifier selected and calibrated by our adaptive approach. The series within figure labeled as *Random Multiple* represents the average performance of all multiple LR raw-data models considered in the random selection process. Similarly,

the series within figure labeled as *Random Binary* represents the average performance of all single metric LR raw-data models considered in the random selection process. And the series labeled *Chance* represents the performance of the classifier that randomly selects the set of modules, which results in the percentage value of the fault-prone classes within a given project. The performance of the CBO classifier produced by our approach clearly outperforms the other three approaches across all investigated projects, except for the Eclipse project for multiple LR model. For Eclipse project, our approach and multiple LR approach achieved the same prediction.

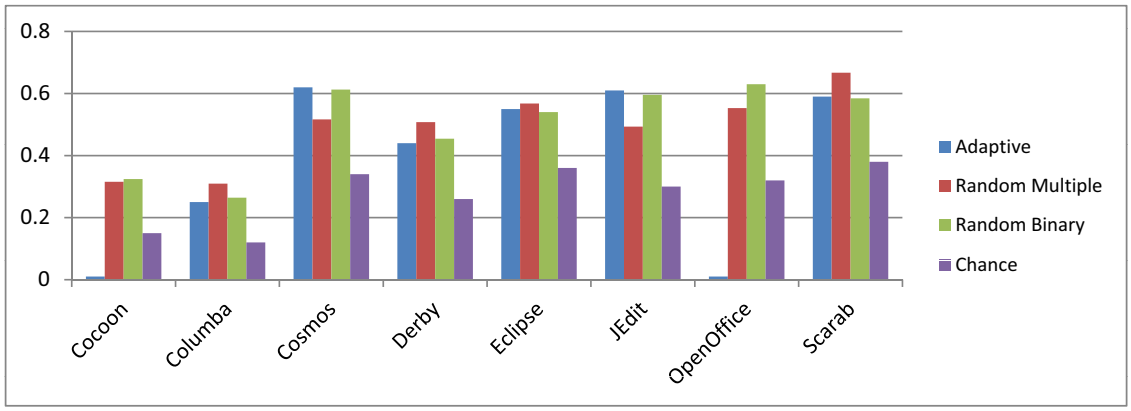


Figure 6.3: LCOM Model F-Scores for Different Fault Classifier Choices

Similarly, Figures 6.2 and 6.3 illustrate side-by-side comparison of the performance of our fault-proneness adaptive prediction approach based on WMC and LCOM metrics respectively, and both are also expressed in models' F-Score values. We observed that WMC models did not do as well as their CBO counterparts, even though they did outperform five out of eighth models generated by randomly selected raw data for both binary and multiple LR. They showed a slight deterioration against two projects using randomly selected raw data models (binary and multiple LR): Eclipse and OpenOffice. Additionally, the multiple LR model outperformed our approach for Columba project. WMC models did significantly better when compared to selection made by chance across all projects. And as discussed in previous chapter, LCOM

metric showed mixed results. While two of our calibrated classifiers experienced a complete collapse, we did still observe an improvement over four different projects, Cosmos, Eclipse, JEdit, and SCarab when compared to a single metric raw-data model. Besides for Cosmos and JEdit projects, multiple LR models outperformed our approach.

Therefore, our results suggest that even though applying the models across systems is far from straightforward, using the approach presented in our research, the model from one project can in fact be helpful at focusing verification and testing effort on fault-prone classes belonging to the other, unrelated project. That is in particular true for CBO and WMC metric. However, logistic regression model performed very well for LCOM model. As set in the objective and dissertation evaluation criteria in Section 1.3, in general terms, the fault-proneness class ranking clearly performs better than chance and also performs at a higher degree than a randomly selected raw-data model developed from a single or multiple metrics.

## 6.2 Implications

The results of our research strongly suggest that complexity measures used in this dissertation can indeed be successfully used to predict fault-prone class modules across seemingly unrelated projects. Using our approach, organizations can leverage fault history data to build reasonable predictors which are likely to be applicable across software systems. And as modern software development produces an abundance of process and product measures along with their fault histories, systematic empirical investigation of this data will provide guidance in several software engineering decisions, and further strengthen the empirical body of knowledge in software engineering.

Furthermore, rather than solely predicting fault-proneness of class modules based on OO metrics, the adaptive approach demonstrated in this work can be adapted to

arbitrary measures of software quality. For example, our measures might involve the cost or severity of investigated faults, but also risk considerations, development costs, or maintenance costs. The underlying idea of developing classifiers, however, remains identical: from the earlier history, we select the appropriate set of metrics which best predicts the future, and develop prediction models in a step-by-step manner as we did in this dissertation. Hence, we show how to systematically build predictors for arbitrary system using fault history data.

## 6.3 Threats to Validity

In this dissertation, we have reported our experience with eight different open-source projects of varying goal, purpose, and domain. Although we could derive successful fault-proneness predictors from the failure history in each of the projects, this may not necessarily generalize to other projects. The work in this dissertation certainly suffers from the project selection bias [32]. Unfortunately, the target application selection process was limited to open-source domain projects implemented in Java programming language. It may not be possible to extend the findings of this study involving open-source software systems to proprietary software due to the different development practices adopted [40]. Further validations with both open-source and proprietary software systems are necessary to help us draw stronger conclusions [7].

Additionally, we have no way of verifying the quality and completeness of the history logs obtained through the SCM system. Even though projects that have been selected seem to have a good quality change log entries, our heuristic for identifying faults for individual modules is far from perfect. It is quite possible that our heuristic fault identification method has not accurately identified every single type of fault present across eight different projects. Manually inspecting change logs containing tens of thousands of lines of text in order to extract fault information is surely bound

to produce errors. We might have identified some log entries as faults, where in actuality they were not faults at all. Or to the contrary, there might be some log entries our heuristic has omitted, while these were in actuality very much representing revisions to faulty modules.

## 6.4 Towards the Framework for Fault Prediction

In order to be widely adopted, suggested fault-prediction techniques should be easy-to-use and applicable across different domains. Additionally, prediction models obtained using those techniques should be simple and intuitive enough to be easily understood and interpreted by developers [28]. As a step toward achieving this goal, in this section we present a high-level framework design to support automation of processes described in this dissertation.

The proposed framework supports the four basic processes shown in Figure 6.4: (1) *Select Metrics*, (2) *Adapt Metrics*, (3) *Create Model*, and (4) *Predict*.

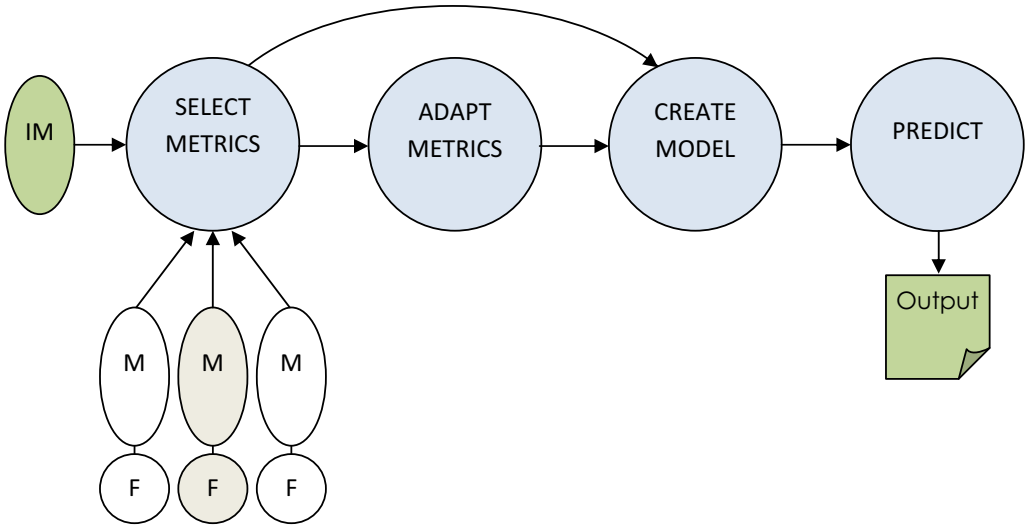


Figure 6.4: High-Level Fault-Prediction Framework Processes

***Select Metrics:*** This framework process is responsible for accepting a project as an input for which the history data is not available, and for which we are interested computing the class fault-proneness probability. Considering that generating predictive values for project's individual modules does not involve software code, framework allows the project to be inputted as source code, but also as the collection of its metric values in case of the intellectual property concerns. The input project within the Figure 6.4 is represented as the *IM* shape. Regardless of the input selection method, we encounter problems related to software measurements that need addressing. In case of metrics, it compels a standardized naming convention. Some metrics, even though identical in terms of their measurement, could possibly use inconsistent naming conventions. Other metrics, on the other hand, might follow the same naming convention, but in fact measure two slight variations of a system's attribute or even quite different attributes altogether. In case of an input as a source code, the framework has to include a mechanism for calculating its metrics. We could certainly allow framework to integrate a third party software tool for metric extraction. However, computing metrics must consider nothing else but its rigorous definition, leaving nothing to interpretation. Presently, there are number of available tools that generate metric values for OO software. How do we ensure that all tools generate the same values for the same metrics? In this dissertation, in Section 3.3 for example, we used the third party tool named Understand to extract CK metrics. Even though this particular issue is outside the scope of this study, such inconsistencies could affect the performance of developed prediction models significantly.

The *Select Process* component of the framework also interacts with the uploadable repository containing projects for which the fault history data is available. Such projects are represented by the combination of $M$ and $F$ shapes within the the Figure 6.4, and will be used as training datasets for the development of our prediction models.

Again, as with our input project without known fault history data, the framework should handle inputting projects with known history data as source code or metric values. Furthermore, history data must also be inputed alongside with the project, and it's complemented with internal extraction and the fault mapping process as presented in this dissertation in Section 3.2.

And finally, the component must also include the statistical inference algorithm that compares our input project metric distribution against every project within the repository for which the fault history data is known. To support the work presented in this dissertation, the framework would have to include a Levene's Test algorithm (Equation 5.1) for the equality of variances presented in Section 5.3 in order to identify a single project most similar to our input project (a single shaded shape identified with M and F within the Figure 6.4), which will be used in the development of our prediction model. Following this process, the flow within the framework can go either to the *Adapt Metrics* process component, or bypassing it (as in case with our LCOM metric, which showed significant decay in classification of faulty classes after the power transformations) to go directly to *Create Model* component.

**Adapt Metrics:** This framework process component is responsible for calibrating metrics data values for both, our input project designated as *IM* in Figure 6.4 and the metrics of the project residing in our metric and fault history repository identified as the most similar to our input project during the *Select Metrics* process. *Adapt Metrics* process component is relatively easy to implement. We need a mechanism for developers to be able to input new and verified transformation techniques that have been proven to improve the validity of statistical measurements. To support the study presented in this dissertation in Section 5.4, we would only need to implement a simple power transformation algorithm as discussed in the previous chapter.

**Create Model:** Within this component, we must implement statistical regression algorithms used to determine the correlation between the chosen metric values of each individual module from the project selected in *Select Metrics* process component and its respective fault-proneness measures. For the framework to support our work, this process component would have to include a logistic regression algorithm presented by Equation 2.2, to arrive at the appropriate values of a metric coefficient and a constant value used to identify fault-proneness probabilities across all input project modules.

**Predict:** Once the fault-proneness model is developed, using the values of project's metrics, this process component calculates the estimated probability values for each module belonging to a project as presented by Equation 4.5. Fault-proneness probability values for each individual module are piped into an accessible output file used by developers as viable decision making tools for the resource allocation during validation activities.

# CHAPTER 7

# CONCLUSION

This chapter provides a summary of the research presented in this dissertation, and discusses future directions in the areas of *software measurement and fault prediction*. In the summary, we present the overview of the conducted work, the results of our empirical study, and the discussion of the extent to which our proposed solution satisfied the research goal. The future work section describes several topics concentrating on complexity design measures, project and data diversity, measures of project similarity and approach automation. This final chapter contains both research summary and possible future work.

## 7.1 Research Summary

One of the main objectives of this dissertation was to propose an approach assisting developers to use fault-proneness models, based on design measurements, as viable decision making tools when applied from one object-oriented system to the other. The identified research problem was divided into the following sub-problems: 1. Devise a strategy that facilitates identifying similar projects; 2. Formulate an approach that adapts or transforms datasets used in development of existing prediction models in order to improve their predictive ability to identify fault-prone modules across different projects, independent of the domain used in the derivation of the predictive model; and 3. Propose a high-level design methodology to support the implementation of the strategy devised in (1.) and the adaptive approach formulated in (2.).

The solution to the first sub-problem included identifying projects that share the same heterogeneous metric distribution across comprising modules. We were interested in calculating the probability that the two independent data samples belong to

the same population. However, since our datasets did not belong to a normally distributed population, we investigated a plethora of alternative, nonparametric equality of variance techniques. At the end of the investigative process, we chose the nonparametric Levene's Test as the most robust and appropriate for the type of skewed data used in this dissertation. The Levene's Test statistic and model selection process is presented in detail in Section 5.3.

Even though the systems used in this work stem from the same open-source development environment, the distributions of metric measures vary across different projects and thus affect the applicability of fault-proneness models, as demonstrated in Section 5.1. The investigation into the second sub-problem consisted of further reducing variability and promoting equality of spread among the datasets through the use of data transformation techniques. Our research into data transformation techniques has led us to a selection of power transformations. They were used to stabilize variance, make the data more normal distribution-like, and improve the validity of measures.

We then used the model selection process identified as a solution to the first subproblem and power transformation techniques to transform data. We transform the training dataset project that lends itself for the development of the prediction model. Additionally, we transform the testing dataset project which acts as the unrelated project for which the fault history data is not available. Only then do we generate the adapted prediction model from the transformed data making it relevant to unrelated project.

Our results suggest that even though applying the models across systems is far from straightforward, by using the approach presented in this work, the model from one project can in fact be helpful at focusing verification and testing effort on fault-prone classes of the other system. As required in the objective and dissertation

evaluation criteria set in Section 1.3 and evaluation of the extent to which the research goal has been accomplished presented in Section 6.1, in general terms, the fault-proneness class ranking clearly performs better than chance and in large part also outperforms a randomly selected raw-data model.

A third contribution of the paper is to propose of the high-level fault-prediction framework, which is only the first step toward the integrated environment that supports automated approach to fault prediction. Furthermore, rather than solely predicting fault-proneness of class modules based on OO metrics, the adaptive approach demonstrated in this work can be adapted to arbitrary measures of software quality. For example, our measures might involve the cost or severity of investigated faults, but also risk considerations, development costs, or maintenance costs. Hence, we show how to systematically build predictors for arbitrary system using fault history data.

## 7.2 Future Work

The research presented in this dissertation provides the foundation for investigating practical applicability of the existing predictive techniques on projects for which the prior fault history data is not known. Presently, the design complexity measures used in this dissertation are rather simple in nature. While CBO turned out to be an overall good fit for our fault prediction, in this dissertation, we solely consider product metrics. We plan to expand the failure data investigation to proprietary software systems due to the different development practices used during their development, and particularly by including more sophisticated product measures [4], as well as process and resource metrics to complement our product measures [59].

A fundamental step in the adaptive approach presented in this work involves the selection process, during which a relevant model from the existing model set is selected

for the new project at hand. We accomplish the selection process by using the test of equality of variances between two projects. Again, we define similar projects by comparing the properties and distributions of their product metrics. In the future, we would like to include data on the process and domain characteristics [38], and consequently determine which process features correlate with the quality of software systems.

In addition to the regression, machine learning approaches have been used in fault prediction. Machine learning approaches are inherently different from regression, raising our interest to evaluate the results of these methods. Thus, in our future research, we would like to investigate and empirically validate the results of several machine learning methods [29, 53], and compare them with the results obtained in this work.

And finally, in order to be widely adopted, suggested fault-prediction techniques should be easy-to-use and prediction models obtained using those techniques should be simple and intuitive enough to be easily understood and interpreted by developers. As a step toward achieving that goal, the extraction and the fault mapping process presented in this dissertation have been automated. However, we still rely on a third party statistical software to manually develop prediction models. In the future, we plan to implement statistical algorithms directly into the framework design, and ultimately into development environments, supporting the decisions of programmers and managers.

# APPENDICES

## A Tables

### A1 Collinearity Analysis using Bivariate Spearman Interrelationships between CK Metrics for Cacoon, Columba, Cosmos, and Derby

Table A1: Bivariate Spearman Interrelationships between CK Metrics for Cacoon, Columba, Cosmos, and Derby

| CBO | NOC | WMC | RFC | DIT | LCOM | Cocoon |
|---|---|---|---|---|---|---|
| 1 | -0.083** | 0.299** | 0.320** | 0.311** | 0.265** | CBO |
| | 1 | -0.038** | -0.151** | -0.120** | 0.067** | NOC |
| | | 1 | 0.673** | -0.003 | 0.638** | WMC |
| | | | 1 | 0.523** | 0.423** | RFC |
| | | | | 1 | 0.058 | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | Columba |
| 1 | -0.306** | 0.341** | 0.409** | 0.317** | 0.374** | CBO |
| | 1 | 0.081** | -0.316** | -0.352** | -0.147** | NOC |
| | | 1 | 0.280** | -0.029 | 0.558** | WMC |
| | | | 1 | 0.873** | 0.330** | RFC |
| | | | | 1 | 0.143** | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | Cosmos |
| 1 | -0.097** | 0.451** | 0.454** | 0.273** | 0.272** | CBO |
| | 1 | 0.007 | 0.022 | -0.285** | -0.112** | NOC |
| | | 1 | 0.808** | 0.203** | 0.504** | WMC |
| | | | 1 | 0.524** | 0.434** | RFC |
| | | | | 1 | 0.196** | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | Derby |
| 1 | 0.062* | 0.549** | 0.555** | 0.262** | 0.383** | CBO |
| | 1 | 0.129** | 0.071 | -0.173** | 0.080 | NOC |
| | | 1 | 0.707** | 0.186** | 0.616** | WMC |
| | | | 1 | 0.676** | 0.395** | RFC |
| | | | | 1 | 0.065 | DIT |
| | | | | | 1 | LCOM |
| Legend: | **correlation is significant at the 0.01 level (2-tailed); | | | | | |
| | *correlation is significant at the 0.05 level (2-tailed); | | | | | |

## A2 Collinearity Analysis using Bivariate Spearman Interrelationships between CK Metrics for Eclipse, JEdit, OpenOffice, and Scarab

Table A2: Bivariate Spearman Interrelationships between CK Metrics for Eclipse, JEdit, OpenOffice, and Scarab

| CBO | NOC | WMC | RFC | DIT | LCOM | Eclipse |
|---|---|---|---|---|---|---|
| 1 | -0.141** | 0.622** | 0.519** | 0.217** | 0.414** | CBO |
| | 1 | 0.017 | -0.034* | -0.168** | -0.070** | NOC |
| | | 1 | 0.622** | 0.042* | 0.643** | WMC |
| | | | 1 | 0.609** | 0.344** | RFC |
| | | | | 1 | 0.045** | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | JEdit |
| 1 | -0.173** | 0.381** | 0.363** | 0.247** | 0.324** | CBO |
| | 1 | 0.109 | 0.053 | -0.179** | 0.044 | NOC |
| | | 1 | 0.901** | 0.252** | 0.745** | WMC |
| | | | 1 | 0.433** | 0.684** | RFC |
| | | | | 1 | 0.225** | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | OpenOffice |
| 1 | -0.057 | 0.382** | 0.399** | 0.251** | 0.326** | CBO |
| | 1 | -0.028 | -0.010 | -0.177** | -0.057 | NOC |
| | | 1 | 0.737** | 0.141 | 0.631** | WMC |
| | | | 1 | 0.579** | 0.454** | RFC |
| | | | | 1 | 0.135 | DIT |
| | | | | | 1 | LCOM |
| CBO | NOC | WMC | RFC | DIT | LCOM | Scarab |
| 1 | 0.007 | 0.570** | 0.538** | 0.206** | 0.324** | CBO** |
| | 1 | 0.170** | 0.184** | -0.066 | 0.149** | NOC |
| | | 1 | 0.679** | -0.129** | 0.679** | WMC |
| | | | 1 | 0.411** | 0.388** | RFC |
| | | | | 1 | -0.246** | DIT |
| | | | | | 1 | LCOM |
| Legend: | **correlation is significant at the 0.01 level (2-tailed); | | | | | |
| | *correlation is significant at the 0.05 level (2-tailed); | | | | | |

## A3 Binary Logistic Regression Results on Metrics versus Faults for Cocoon, Columba, Cosmos, and Derby

Table A3: Logistic Regression for Cocoon, Columba, Cosmos, and Derby

| Cocoon | CBO | NOC | WMC | RFC | DIT | LCOM |
|---|---|---|---|---|---|---|
| Coeff. | 0.193 | - | 0.075 | 0.035 | 0.120 | 0.016 |
| Const. | -0.603 | - | -0.530 | -0.467 | -0.294 | -0.653 |
| p-value | 0.000 | 0.472 | 0.000 | 0.000 | 0.034 | 0.000 |
| HL | 12.683 | - | 15.218 | 17.291 | 10.737 | 19.016 |
| DF | 5 | - | 8 | 8 | 5 | 6 |
| p-value | 0.027 | - | 0.055 | 0.027 | 0.057 | 0.004 |
| Columba | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.096 | - | 0.058 | - | 0.131 | 0.014 |
| Const. | -1.208 | - | -0.384 | - | -0.322 | -0.507 |
| p-value | 0.000 | 0.735 | 0.000 | 0.099 | 0.015 | 0.000 |
| HL | 14.477 | - | 3.315 | - | 25.148 | 1.391 |
| DF | 8 | - | 6 | - | 4 | 4 |
| p-value | 0.070 | - | 0.768 | - | 0.000 | 0.846 |
| Cosmos | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.139 | - | 0.129 | 0.093 | 0.704 | 0.010 |
| Const. | -0.495 | - | -0.825 | -0.996 | -1.135 | -0.402 |
| p-value | 0.000 | 1.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| HL | 7.545 | - | 27.697 | 34.394 | 13.907 | 25.130 |
| DF | 5 | - | 8 | 7 | 3 | 5 |
| p-value | 0.183 | - | 0.001 | 0.000 | 0.003 | 0.000 |
| Derby | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.092 | - | 0.064 | 0.008 | - | 0.015 |
| Const. | -0.921 | - | -0.871 | -0.416 | -0.530 | -0.738 |
| p-value | 0.000 | 0.062 | 0.000 | 0.000 | 0.072 | 0.000 |
| HL | 21.620 | - | 23.179 | 51.881 | - | 28.031 |
| DF | 8 | - | 8 | 8 | - | 6 |
| p-value | 0.006 | - | 0.003 | 0.000 | - | 0.000 |
| Legend: | HL = Hosmer-Lemeshow chi-square; | | | | | |
| | DF = Hosmer-Lemeshow degrees of freedom; | | | | | |

## A4 Binary Logistic Regression Results on Metrics versus Faults for Eclipse, JEdit, OpenOffice, and Scarab

Table A4: Logistic Regression for Eclipse, JEdit, OpenOffice, and Scarab

| Eclipse | CBO | NOC | WMC | RFC | DIT | LCOM |
|---|---|---|---|---|---|---|
| Coeff. | 0.081 | - | 0.055 | 0.006 | 0.272 | 0.015 |
| Const. | -0.687 | - | -0.585 | -0.205 | -0.575 | -0.711 |
| p-value | 0.000 | 0.152 | 0.000 | 0.000 | 0.000 | 0.000 |
| HL | 28.119 | - | 33.356 | 115.393 | 22.702 | 17.698 |
| DF | 8 | - | 8 | 8 | 3 | 6 |
| p-value | 0.000 | - | 0.000 | 0.000 | 0.000 | 0.007 |
| Jedit | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.255 | - | 0.119 | 0.090 | 0.635 | 0.031 |
| Const. | -0.931 | - | -0.689 | -0.789 | -1.050 | -1.084 |
| p-value | 0.000 | 0.699 | 0.000 | 0.000 | 0.001 | 0.000 |
| HL | 4.474 | - | 29.357 | 37.923 | 12.372 | 2.182 |
| DF | 6 | - | 6 | 7 | 2 | 4 |
| p-value | 0.613 | - | 0.000 | 0.000 | 0.002 | 0.702 |
| OpenOffice | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.134 | -0.621 | 0.050 | - | - | 0.014 |
| Const. | -0.683 | 0.133 | -0.395 | - | - | -0.531 |
| p-value | 0.000 | 0.019 | 0.030 | 0.320 | 0.799 | 0.001 |
| HL | 12.152 | 0.000 | 22.766 | - | - | 9.272 |
| DF | 7 | 0 | 8 | - | - | 5 |
| p-value | 0.096 | - | 0.004 | - | - | 0.099 |
| Scarab | CBO | NOC | WMC | RFC | DIT | LCOM |
| Coeff. | 0.345 | - | 0.204 | 0.102 | 0.517 | 0.020 |
| Const. | -1.651 | - | -1.393 | -1.460 | -1.072 | -0.677 |
| p-value | 0.000 | 0.213 | 0.000 | 0.000 | 0.004 | 0.000 |
| HL | 3.027 | - | 7.741 | 19.269 | 0.881 | 10.273 |
| DF | 6 | - | 7 | 8 | 2 | 4 |
| p-value | 0.805 | - | 0.356 | 0.013 | 0.644 | 0.036 |
| Legend: | HL = Hosmer-Lemeshow chi-square value; | | | | | |
| | DF = Hosmer-Lemeshow degrees of freedom; | | | | | |

## A5 Multiple Logistic Regression Results on Metrics versus Faults across All Projects

Table A5: Multiple Logistic Regression

| Cocoon | Const. | CBO | WMC | LCOM | Columba | Const. | CBO | WMC | LCOM |
|---|---|---|---|---|---|---|---|---|---|
| Coeff. | -0.990 | 0.165 | - | 0.011 | Coeff. | -1.208 | 0.096 | - | - |
| p-value | 0 | 0 | 0.056 | 0 | p-value | 0 | 0 | 0.473 | 0.301 |
| | *Goodness of fit test* | | | | | *Goodness of fit test* | | | |
| | HL: 4.562 | | | | | HL: 14.477 | | | |
| | DF: 8 | | | | | DF: 8 | | | |
| | p-value: 0.803 | | | | | p-value: 0.800 | | | |
| Cosmos | Const. | CBO | WMC | LCOM | Derby | Const. | CBO | WMC | LCOM |
| Coeff. | -1.021 | 0.101 | 0.104 | - | Coeff. | -1.181 | 0.062 | 0.041 | - |
| p-value | 0 | 0 | 0 | 0.489 | p-value | 0 | 0 | 0 | 0.268 |
| | *Goodness of fit test* | | | | | *Goodness of fit test* | | | |
| | HL: 36.399 | | | | | HL: 28.713 | | | |
| | DF: 8 | | | | | DF: 8 | | | |
| | p-value: 0 | | | | | p-value: 0 | | | |
| Eclipse | Const. | CBO | WMC | LCOM | Jedit | Const. | CBO | WMC | LCOM |
| Coeff. | -1.008 | 0.066 | - | 0.009 | Coeff. | -1.537 | 0.187 | - | 0.025 |
| p-value | 0 | 0 | 0.119 | 0 | p-value | 0 | 0.003 | 0.547 | 0 |
| | *Goodness of fit test* | | | | | *Goodness of fit test* | | | |
| | HL: 28.106 | | | | | HL: 9.213 | | | |
| | DF: 8 | | | | | DF: 8 | | | |
| | p-value: 0 | | | | | p-value: 0.325 | | | |
| OpenOffice | Const. | CBO | WMC | LCOM | Scarab | Const. | CBO | WMC | LCOM |
| Coeff. | -0.955 | 0.119 | - | 0.009 | Coeff. | -1.990 | 0.281 | 0.100 | - |
| p-value | 0.000 | 0.000 | 0.295 | 0.040 | p-value | 0.000 | 0.000 | 0.010 | 0.467 |
| | *Goodness of fit test* | | | | | *Goodness of fit test* | | | |
| | HL: 11.079 | | | | | HL: 2.605 | | | |
| | DF: 8 | | | | | DF: 8 | | | |
| | p-value: 0.197 | | | | | p-value: 0.967 | | | |
| Legend: | HL = Hosmer-Lemeshow chi-square; | | | | | | | | |
| | DF = Hosmer-Lemeshow degrees of freedom; | | | | | | | | |

## A6 F-Scores for Cross-System Logistic Models Validations across Different Projects

Table A6: F-scores for All Models across All Projects

| Model: CBO | | | | Testing Dataset | | | | |
|---|---|---|---|---|---|---|---|---|
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | **0.33*** | 0.27 | 0.58 | 0.51 | 0.58 | 0.50 | 0.66 | 0.78 |
| Columba | 0.15 | **0.37*** | 0.24 | 0.48 | 0.50 | 0.24 | 0.31 | 0.47 |
| Cosmos | 0.33 | 0.27 | **0.58*** | 0.51 | 0.58 | 0.50 | 0.66 | 0.78 |
| Derby | 0.22 | 0.34 | 0.34 | **0.51*** | 0.55 | 0.34 | 0.44 | 0.62 |
| Eclipse | 0.22 | 0.32 | 0.37 | 0.51 | **0.56*** | 0.39 | 0.47 | 0.65 |
| JEdit | 0.33 | 0.27 | 0.58 | 0.51 | 0.58 | **0.50*** | 0.66 | 0.78 |
| OpenOffice | 0.32 | 0.29 | 0.49 | 0.51 | 0.58 | 0.49 | **0.58*** | 0.73 |
| Scarab | 0.34 | 0.28 | 0.54 | 0.51 | 0.57 | 0.49 | 0.58 | **0.76*** |
| Model: WMC | | | | Testing Dataset | | | | |
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | **0.37*** | 0.25 | 0.59 | 0.52 | 0.56 | 0.51 | 0.45 | 0.59 |
| Columba | 0.36 | **0.25*** | 0.64 | 0.51 | 0.57 | 0.55 | 0.51 | 0.62 |
| Cosmos | 0.36 | 0.25 | **0.64*** | 0.51 | 0.57 | 0.55 | 0.51 | 0.62 |
| Derby | 0.27 | 0.22 | 0.30 | **0.53*** | 0.45 | 0.34 | 0.26 | 0.43 |
| Eclipse | 0.33 | 0.24 | 0.42 | 0.53 | **0.50*** | 0.43 | 0.36 | 0.50 |
| JEdit | 0.36 | 0.24 | 0.67 | 0.49 | 0.58 | **0.56*** | 0.59 | 0.64 |
| OpenOffice | 0.37 | 0.24 | 0.59 | 0.52 | 0.56 | 0.51 | **0.45*** | 0.59 |
| Scarab | 0.36 | 0.25 | 0.64 | 0.51 | 0.57 | 0.55 | 0.51 | **0.62*** |
| Model: LCOM | | | | Testing Dataset | | | | |
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | **0.32*** | 0.27 | 0.61 | 0.46 | 0.54 | 0.59 | 0.63 | 0.58 |
| Columba | 0.32 | **0.26*** | 0.61 | 0.45 | 0.54 | 0.60 | 0.63 | 0.59 |
| Cosmos | 0.32 | 0.27 | **0.61*** | 0.46 | 0.54 | 0.59 | 0.63 | 0.59 |
| Derby | 0.33 | 0.27 | 0.61 | **0.45*** | 0.54 | 0.59 | 0.63 | 0.58 |
| Eclipse | 0.33 | 0.27 | 0.61 | 0.46 | **0.54*** | 0.59 | 0.63 | 0.58 |
| JEdit | 0.32 | 0.26 | 0.62 | 0.45 | 0.54 | **0.61*** | 0.63 | 0.59 |
| OpenOffice | 0.33 | 0.26 | 0.61 | 0.45 | 0.54 | 0.60 | **0.63*** | 0.58 |
| Scarab | 0.32 | 0.25 | 0.62 | 0.45 | 0.54 | 0.61 | 0.63 | **0.59*** |
| Model: Multiple variable | | | | Testing Dataset | | | | |
| Training Dataset | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOffice | Scarab |
| Cocoon | **0.34*** | 0.28 | 0.67 | 0.49 | 0.59 | 0.60 | 0.67 | 0.76 |
| Columba | 0.15 | **0.37*** | 0.24 | 0.48 | 0.50 | 0.24 | 0.31 | 0.47 |
| Cosmos | 0.36 | 0.28 | **0.65*** | 0.51 | 0.59 | 0.57 | 0.62 | 0.75 |
| Derby | 0.28 | 0.36 | 0.30 | **0.55*** | 0.54 | 0.35 | 0.38 | 0.52 |
| Eclipse | 0.36 | 0.32 | 0.47 | 0.52 | **0.58*** | 0.47 | 0.57 | 0.65 |
| Jedit | 0.33 | 0.28 | 0.65 | 0.48 | 0.57 | **0.62*** | 0.67 | 0.73 |
| OpenOffice | 0.35 | 0.29 | 0.58 | 0.51 | 0.59 | 0.58 | **0.61*** | 0.74 |
| Scarab | 0.36 | 0.29 | 0.57 | 0.52 | 0.59 | 0.52 | 0.59 | **0.73*** |
| Legend: *threshold value, where training and testing dataset belong to the same project; | | | | | | | | |

## A7 Descriptive Statistics for CBO, WMC, and LCOM across Different Projects

Table A7: Descriptive Statistics for CBO, WMC, and LCOM across All Projects

| CBO | Cocoon | Columba | Cosmos | Derby | Eclipse | Jedit | OpenOffice | Scarab |
|---|---|---|---|---|---|---|---|---|
| Max | 78 | 76 | 57 | 126 | 210 | 56 | 48 | 68 |
| 3rd Quartile | 4 | 14 | 5 | 12 | 11 | 5 | 6 | 7 |
| Median | 2 | 7 | 2 | 5 | 5 | 2 | 3 | 2 |
| 1st Quartile | 0 | 3 | 0 | 2 | 1 | 1 | 1 | 0 |
| Min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| WMC | Cocoon | Columba | Cosmos | Derby | Eclipse | Jedit | OpenOffice | Scarab |
| Max | 63 | 71 | 82 | 308 | 405 | 213 | 131 | 166 |
| 3rd Quartile | 7 | 7 | 9 | 15 | 13 | 6 | 9 | 8 |
| Median | 4 | 4 | 5 | 7 | 7 | 3 | 5 | 3 |
| 1st Quartile | 2 | 2 | 2 | 3 | 3 | 1 | 2 | 1 |
| Min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LCOM | Cocoon | Columba | Cosmos | Derby | Eclipse | Jedit | OpenOffice | Scarab |
| Max | 100 | 100 | 100 | 100 | 100 | 100 | 100 | 100 |
| 3rd Quartile | 71 | 65 | 76 | 79 | 81 | 62 | 70 | 66 |
| Median | 33 | 0 | 33 | 50 | 50 | 0 | 28 | 0 |
| 1st Quartile | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Min | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## A8 Levene's Test Results with its $F$ Statistics and *P-values* for Metric Equality of Variances across All Projects

Table A8: Levene's Test Results for Equality of Variances

| Metric: CBO | Stat. | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOff. | Scarab |
|---|---|---|---|---|---|---|---|---|---|
| Columba | F | 2.642 | 0 | | | | | | |
| | p-value | 0.104 | 1 | | | | | | |
| Cosmos | F | 16.318 | 9.307 | 0 | | | | | |
| | p-value | 0 | 0.002 | 1 | | | | | |
| Derby | F | 46.388 | 42.981 | 5.569 | 0 | | | | |
| | p-value | 0 | 0 | 0.018 | 1 | | | | |
| Eclipse | F | 101.567 | 53.204 | 16.511 | 1.414 | 0 | | | |
| | p-value | 0 | 0 | 0 | 0.234 | 1 | | | |
| JEdit | F | 24.369 | 54.621 | 68.796 | 92.77 | 105.62 | 0 | | |
| | p-value | 0 | 0 | 0 | 0 | 0 | 1 | | |
| OpenOffice | F | 0.008 | 14.288 | 10.971 | 35.872 | 34.053 | 12.856 | 0 | |
| | p-value | 0.930 | 0 | 0.001 | 0 | 0 | 0 | 1 | |
| Scarab | F | 32.254 | 0.727 | 8.159 | 0.603 | 0.004 | 68.275 | 20.761 | 0 |
| | p-value | 0 | 0.394 | 0.004 | 0.438 | 0.947 | 0 | 0 | 1 |
| Metric: WMC | Stat. | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOff. | Scarab |
| Columba | F | 47.142 | 0 | | | | | | |
| | p-value | 0 | 1 | | | | | | |
| Cosmos | F | 1.807 | 47.457 | 0 | | | | | |
| | p-value | 0.179 | 0 | 1 | | | | | |
| Derby | F | 5.527 | 30.459 | 9.496 | 0 | | | | |
| | p-value | 0.019 | 0 | 0.002 | 1 | | | | |
| Eclipse | F | 16.899 | 82.123 | 14.389 | 0.836 | 0 | | | |
| | p-value | 0 | 0 | 0 | 0.361 | 1 | | | |
| JEdit | F | 0.346 | 24.357 | 3.267 | 16.791 | 22.947 | 0 | | |
| | p-value | 0.557 | 0 | 0.071 | 0 | 0 | 1 | | |
| OpenOffice | F | 4.348 | 2.043 | 4.939 | 18.181 | 20.122 | 3.364 | 0 | |
| | p-value | 0.037 | 0.153 | 0.026 | 0 | 0 | 0.067 | 1 | |
| Scarab | F | 27.374 | 87.966 | 13.232 | 0.068 | 0.628 | 24.587 | 26.524 | 0 |
| | p-value | 0 | 0 | 0 | 0.795 | 0.428 | 0 | 0 | 1 |
| Metric: LCOM | Stat. | Cocoon | Columba | Cosmos | Derby | Eclipse | JEdit | OpenOff. | Scarab |
| Columba | F | 0.213 | 0 | | | | | | |
| | p-value | 0.644 | 1 | | | | | | |
| Cosmos | F | 66.516 | 28.541 | 0 | | | | | |
| | p-value | 0 | 0 | 1 | | | | | |
| Derby | F | 13.66 | 8.109 | 12.033 | 0 | | | | |
| | p-value | 0 | 0.004 | 0.001 | 1 | | | | |
| Eclipse | F | 35.21 | 34.462 | 7.031 | 0.265 | 0 | | | |
| | p-value | 0 | 0 | 0.008 | 0.607 | 1 | | | |
| JEdit | F | 5.057 | 1.126 | 40.182 | 19.265 | 45.127 | 0 | | |
| | p-value | 0.025 | 0.289 | 0 | 0 | 0 | 1 | | |
| OpenOffice | F | 3.434 | 1.572 | 15.863 | 2.343 | 9.254 | 5.078 | 0 | |
| | p-value | 0.064 | 0.21 | 0 | 0.126 | 0.002 | 0.024 | 1 | |
| Scarab | F | 0.633 | 0.18 | 15.729 | 5.005 | 10.839 | 0.153 | 1.216 | 0 |
| | p-value | 0.426 | 0.672 | 0 | 0.025 | 0.001 | 0.696 | 0.27 | 1 |

## A9 Logistic Regression on Transformed Datasets

Table A9: Logistic Regression on Transformed Datasets

| Project: | | Cocoon | | Project: | | Columba | |
|---|---|---|---|---|---|---|---|
| Model from: | OpenOffice | JEdit | Columba | Model from: | Scarab | OpenOffice | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Coeff. | 1.761 | 3.263 | 0.523 | Coeff. | 3.425 | 1.804 | 0.660 |
| Const. | -1.098 | -2.597 | -1.301 | Const. | -3.596 | -1.646 | -0.581 |
| p-value | 0.000 | 0.000 | 0.000 | p-value | 0.000 | 0.000 | 0.000 |
| HL | 14.491 | 18.385 | 5.253 | HL | 14.821 | 16.238 | 19.650 |
| p-value | 0.043 | 0.005 | 0.262 | p-value | 0.022 | 0.039 | 0.001 |
| Project: | | Cosmos | | Project: | | Derby | |
| Model from: | Derby | Cocoon | Eclipse | Model from: | Scarab | Scarab | Eclipse |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Coeff. | 2.346 | 1.827 | 0.575 | Coeff. | 3.425 | 3.457 | 0.575 |
| Const. | -2.038 | -1.536 | -0.707 | Const. | -3.168 | -3.693 | -0.707 |
| p-value | 0.000 | 0.000 | 0.000 | p-value | 0.000 | 0.000 | 0.000 |
| HL | 5.005 | 11.331 | 33.672 | HL | 14.821 | 5.460 | 33.672 |
| p-value | 0.757 | 0.184 | 0.000 | p-value | 0.022 | 0.604 | 0.000 |
| Project: | | Eclipse | | Project: | | JEdit | |
| Model from: | Scarab | Scarab | Derby | Model from: | OpenOffice | Cocoon | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Coeff. | 3.425 | 3.457 | 2.346 | Coeff. | 1.761 | 1.827 | 0.660 |
| Const. | -3.168 | -3.693 | -2.038 | Const. | -1.098 | -1.391 | -0.581 |
| p-value | 0.000 | 0.000 | 0.000 | p-value | 0.000 | 0.000 | 0.000 |
| HL | 14.821 | 5.460 | 5.005 | HL | 14.491 | 11.331 | 19.650 |
| p-value | 0.022 | 0.604 | 0.757 | p-value | 0.043 | 0.184 | 0.001 |
| Project: | | OpenOffice | | Project: | | Scarab | |
| Model from: | Coccoon | Scarab | Scarab | Model from: | Eclipse | Derby | JEdit |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Coeff. | 1.978 | 3.457 | 0.660 | Coeff. | 1.941 | 2.584 | 1.197 |
| Const. | -1.185 | -3.261 | -1.547 | Const. | -1.536 | -2.590 | -1.184 |
| p-value | 0.000 | 0.000 | 0.000 | p-value | 0.000 | 0.000 | 0.000 |
| HL | 4.202 | 5.460 | 19.650 | HL | 12.037 | 9.296 | 2.609 |
| p-value | 0.521 | 0.604 | 0.001 | p-value | 0.150 | 0.318 | 0.625 |

## A10 The Recall, Precision, and F-Scores for all Transformed Data Prediction Models across Different Projects

Table A10: Evaluation of Transformed Data Models across Different Projects

| Project: | | Cocoon | | Project: | | Columba | |
|---|---|---|---|---|---|---|---|
| Model from: | OpenOffice | JEdit | Columba | Model from: | Scarab | OpenOffice | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Recall | 61% | 59% | - | Recall | 71% | 44% | 65% |
| Precision | 25% | 26% | - | Precision | 24% | 16% | 16% |
| F-Score | 0.35 | 0.36 | - | F-Score | 0.36 | 0.24 | 0.25 |
| Project: | | Cosmos | | Project: | | Derby | |
| Model from: | Derby | Cocoon | Eclipse | Model from: | Scarab | Scarab | Eclipse |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Recall | 56% | 61% | 63% | Recall | 64% | 62% | 75% |
| Precision | 73% | 73% | 60% | Precision | 42% | 46% | 31% |
| F-Score | 0.63 | 0.67 | 0.62 | F-Score | 0.51 | 0.53 | 0.44 |
| Project: | | Eclipse | | Project: | | JEdit | |
| Model from: | Scarab | Scarab | Derby | Model from: | OpenOffice | Cocoon | Scarab |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Recall | 56% | 48% | 76% | Recall | 67% | 74% | 79% |
| Precision | 58% | 53% | 43% | Precision | 41% | 53% | 50% |
| F-Score | 0.57 | 0.50 | 0.55 | F-Score | 0.51 | 0.62 | 0.61 |
| Project: | | OpenOffice | | Project: | | Scarab | |
| Model from: | Coccoon | Scarab | Scarab | Model from: | Eclipse | Derby | JEdit |
| Used metric: | CBO | WMC | LCOM | Used metric: | CBO | WMC | LCOM |
| Recall | 67% | 38% | - | Recall | 79% | 65% | 59% |
| Precision | 64% | 55% | - | Precision | 79% | 70% | 59% |
| F-Score | 0.66 | 0.45 | - | F-Score | 0.79 | 0.67 | 0.59 |

REFERENCES


[1] Fernando Brito e. Abreu and Walcelio Melo. Evaluating the impact of object-oriented design on software quality. In *METRICS '96: Proceedings of the 3rd International Symposium on Software Metrics*, page 90, Washington, DC, USA, 1996. IEEE Computer Society.

[2] Fernando Brito e Abreu, Gonçalo Pereira, and Pedro Sousa. A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems. In *CSMR '00: Proceedings of the Conference on Software Maintenance and Reengineering*, page 13, Washington, DC, USA, 2000. IEEE Computer Society.

[3] Paul Allison. *Multiple Regression: A Primer*. Pine Forge Press, Thousand Oaks, CA, 1999.

[4] Djuradj Babich, Peter J. Clarke, James F. Power, and B. M. Golam Kibria. Using a class abstraction technique to predict faults in oo classes: a case study through six releases of the eclipse jdt. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 1419–1424, New York, NY, USA, 2011. ACM.

[5] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Trans. Softw. Eng.*, 28(1):4–17, 2002.

[6] Victor R. Basili, Lionel C. Briand, and Walcélio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22(10):751–761, 1996.

[7] Victor R. Basili, Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. *IEEE Trans. Softw. Eng.*, 25(4):456–473, July 1999.

[8] B. Behlendorf, C. M. Pilato, G. Stein, K. Fogel, K. Hancock, and B. Collins-Sussman. Apache subversion project homepage. `http://subversion.apache.org/`, 2006.

[9] B. Berliner. Cvs II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352, Berkeley, CA, 1990. USENIX Association.

[10] Salah Bouktif, Danielle Azar, Doina Precup, Houari Sahraoui, and Balazs Kegl. Improving rule set based software quality prediction: A genetic algorithm-based approach. *Journal of Object Technology*, 3(4):227–241, 2004.

[11] Lionel C. Briand, Walcelio L. Melo, and Jurgen Wust. Assessing the applicability of fault-proneness models across object-oriented software projects. *IEEE Transactions on Software Engineering*, 28:706–720, 2002.

[12] Fernando Brito e Abreu and Rogério Carapuça. Candidate metrics for object-oriented software within a taxonomy framework. *J. Syst. Softw.*, 26:87–96, July 1994.

[13] Morton B. Brown and Alan B. Forsythe. Robust tests for the equality of variances. *Journal of the American Statistical Association*, 69(346):pp. 364–367, 1974.

[14] Manfred Broy, Florian Deissenboeck, and Markus Pizka. Demystifying maintainability. In *WoSQ '06: Proceedings of the 2006 international workshop on Software quality*, pages 21–26, New York, NY, USA, 2006. ACM.

[15] Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering (2nd ed.)*. Pearson Education, Upper Saddle River, NJ, USA, 2004.

[16] Ana Erika Camargo Cruz and Koichiro Ochimizu. Towards logistic regression models for predicting fault-prone code across software projects. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, pages 460–463, Washington, DC, USA, 2009. IEEE Computer Society.

[17] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, 20(6):476–493, 1994.

[18] W. J. Conover. *Practical Nonparametric Statistics (3rd ed.)*. Wiley, New York, NY, USA, 1999.

[19] WJ Conover and R.L. Iman. Rank transformations as a bridge between parametric and nonparametric statistics. *American Statistician*, pages 124–129, 1981.

[20] Kalhed El Emam, Saïda Benlarbi, Nishith Goel, and Shesh N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Trans. Softw. Eng.*, 27(7):630–650, 2001.

[21] Khaled El Emam, Walcelio Melo, and Javam C. Machado. The prediction of faulty classes using object-oriented design metrics. *J. Syst. Softw.*, 56(1):63–75, 2001.

[22] Michael English, Chris Exton, Irene Rigon, and Brendan Cleary. Fault detection and prediction in an open-source software project. In *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, pages 1–11, New York, NY, USA, 2009. ACM.

[23] Letha H. Etzkorn, Sampson E. Gholston, Julie L. Fortune, Cara E. Stein, Dawn Utley, Phillip A. Farrington, and Glenn W. Cox. A comparison of cohesion metrics for object-oriented systems. *Information and Software Technology*, 46(10):677 – 687, 2004.

[24] Norman E. Fenton and Martin Neil. Software metrics: roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 357–370, New York, NY, USA, 2000. ACM.

[25] Norman E. Fenton and Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Trans. Softw. Eng.*, 26(8):797–814, 2000.

[26] Michael Fischer, Martin Pinzger, and Harald Gall. Populating a release history database from version control and bug tracking systems. In *In Proceedings of the International Conference on Software Maintenance*, pages 23–32, 2003.

[27] Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7):653–661, 2000.

[28] A. Güneş Koru and Hongfang Liu. Identifying and characterizing change-prone classes in two large-scale open-source products. *J. Syst. Softw.*, 80:63–73, January 2007.

[29] Lan Guo, Yan Ma, Bojan Cukic, and Harshinder Singh. Robust prediction of fault-proneness by random forests. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE '04, pages 417–428, Washington, DC, USA, 2004. IEEE Computer Society.

[30] Tibor Gyimothy, Rudolf Ferenc, and Istvan Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *IEEE Trans. Softw. Eng.*, 31(10):897–910, 2005.

[31] Mary Jean Harrold. Testing: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 61–72, New York, NY, USA, 2000. ACM.

[32] James Heckman. Sample selection bias as a specification error. *Econometrica*, 47(1):153–161, January 1979.

[33] David C. Hoaglin, Frederick Mosteller, and John W. Tukey (Editor). *Understanding Robust and Exploratory Data Analysis*. Wiley-Interscience, 1 edition, 2000.

[34] David W. Hosmer and Stanley Lemeshow. *Applied Logistic Regression*. John Wiley and Sons, 2nd edition, 2000.

[35] IBM. SPSS. `http://www.spss.com/`, 2009.

[36] Information Systems/Technical Academic Support, Center for Statistical Computing Support, Social Sciences Teaching and Research

Statistics (SSTARS), Univ. of Kentucky. Multicollinearity in logistic regression. http://www.uky.edu/ComputingCenter/SSTARS/MulticollinearityinLogisticRegression.htm, 2006.

[37] Lucas Layman, Gunnar Kudrjavets, and Nachiappan Nagappan. Iterative identification of fault-prone binaries using in-process metrics. In *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 206–212, New York, NY, USA, 2008. ACM.

[38] Lucas Layman, Laurie Williams, and Lynn Cunningham. Exploring extreme programming in context: An industrial case study. In *Proceedings of the Agile Development Conference*, ADC '04, pages 32–41, Washington, DC, USA, 2004. IEEE Computer Society.

[39] J. Scott Long. *Regression models for categorical and limited dependent variables*. Number 7 in Advanced quantitative techniques in the social sciences. Sage Publ., Thousand Oaks, Calif. [u.a.], 1997.

[40] Alan MacCormack, John Rusnak, and Carliss Y. Baldwin. Exploring the structure of complex software designs: An empirical study of open source and proprietary code. *Manage. Sci.*, 52:1015–1030, July 2006.

[41] Rupa Mahanti and Jiju Antony. Confluence of six sigma, simulation and software development. *Managerial Auditing Journal*, 20(7):739–762, 2005.

[42] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[43] Audris Mockus and Lawrence G. Votta. Identifying reasons for software changes using historic databases. In *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ICSM '00, pages 120–, Washington, DC, USA, 2000. IEEE Computer Society.

[44] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *ICSE '06: Proceedings of the 28th international conference on Software engineering*, pages 452–461, New York, NY, USA, 2006. ACM.

[45] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The influence of organizational structure on software quality: an empirical case study. In *ICSE '08: Proceedings of the 30th international conference on Software engineering*, pages 521–530, New York, NY, USA, 2008. ACM.

[46] Hector M. Olague, Letha H. Etzkorn, Sampson Gholston, and Stephen Quattlebaum. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. *IEEE Trans. Softw. Eng.*, 33(6):402–419, 2007.

[47] The Mozilla Organization. Bugzilla. `http://www.bugzilla.org/`, 2006.

[48] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Where the bugs are. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 86–96, New York, NY, USA, 2004. ACM.

[49] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, 2005.

[50] Thomas J. Ostrand, Elaine J. Weyuker, and Robert M. Bell. Automating algorithms for the identification of fault-prone files. In *ISSTA '07: Proceedings of the 2007 international symposium on Software testing and analysis*, pages 219–227, New York, NY, USA, 2007. ACM.

[51] Scientific Toolworks Inc. Understand. `http://www.scitools.com/products/understand/`, 2004.

[52] Dinesh Sharma and B. M. Golam Kibria. On some test statistics for testing homogeneity of variances: a comparative study. *Journal of Statistical Computation and Simulation*, 82(12), 2012.

[53] Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Application of support vector machine to predict fault prone classes. *SIGSOFT Softw. Eng. Notes*, 34(1):1–6, January 2009.

[54] Jacek Śliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 international workshop on Mining software repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.

[55] M. Stone. Cross-validatory choice and assessment of statistical predictions. *Roy. Stat. Soc.*, 36:111–147, 1974.

[56] Ramanath Subramanyam and M. S. Krishnan. Empirical analysis of metrics for object-oriented design complexity: Implications for software defects. *IEEE Trans. Softw. Eng.*, 29(4):297–310, 2003.

[57] Sun Microsystems, Inc. Core Java J2SE 5.0, February 2005. `http://java.sun.com/j2se/1.5.0/index.jsp` (Mar. 2006).

[58] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen. An empirical study on object-oriented metrics. In *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, page 242, Washington, DC, USA, 1999. IEEE Computer Society.

[59] Gabriella Toth, Adám Zoltán Vegh, Arpád Beszedes, and Tibor Gyimothy. Adding process metrics to enhance modification complexity prediction. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 201–204, Washington, DC, USA, 2011. IEEE Computer Society.

[60] Davor Čubranić and Gail C. Murphy. Hipikat: recommending pertinent software development artifacts. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.

[61] Will G. Hopkins. A new view of statistics. `http://www.sportsci.org/resource/stats`, 2003.

[62] Jie Xu, Danny Ho, and Luiz Fernando Capretz. An empirical validation of object-oriented design metrics for fault prediction. *Journal of Computer Science*, 4(7):571–577, 2008.

[63] Ping Yu, Tarja Systä, and Hausi A. Müller. Predicting fault-proneness using oo metrics: An industrial case study. In *CSMR '02: Proceedings of the 6th European Conference on Software Maintenance and Reengineering*, pages 99–107, Washington, DC, USA, 2002. IEEE Computer Society.

[64] Andreas Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[65] Yuming Zhou and Hareton Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Trans. Softw. Eng.*, 32(10):771–789, 2006.

# VITA

## DJURADJ BABIC

| | |
|---|---|
| 2012 | Doctoral Candidate in Computer Science<br>Florida International University, Miami, Florida |
| 2005 | Master of Science in Computer Science<br>Florida International University, Miami, Florida |
| 2003 | Bachelor of Science in Computer Science<br>Florida International University, Miami, Florida |

## PUBLICATIONS AND PRESENTATIONS

Peter J. Clarke, James F. Power, Djuradj Babich, and Tariq M. King: A Testing Strategy for Abstract Classes. Journal of Software Testing, Verification and Reliability 22(3) 2012: 147–169

Peter J. Clarke, Djuradj Babich, Tariq M. King and B.M. Golam Kibria: *Analyzing Clusters of Class Characteristics in OO Applications.* Journal of Systems and Software 81(12) 2008: 2269–2286

Peter J. Clarke, Brian A. Malloy, Junhua Ding and Djuradj Babich: A *Tool to Automatically Map Implementation-Based Testing Techniques to Classes.* Journal of Software Engineering and Knowledge Engineering 16(4) 2006: 585–614

Djuradj Babich, Peter J. Clarke, James F. Power and B. M. Golam Kibria: *Using a Class Abstraction Technique to Predict Faults in OO Classes: A case study through six releases of the Eclipse JDT.* Symposium on Applied Computing 2011: 1419–1424

Peter J. Clarke, James F. Power, Djuradj Babich, and Tariq M. King: *An Approach to Support Intra-Class Testing of Abstract Classes.* International Symposium on Software Reliability Engineering 2007: 191–200

Tariq M. King, Djuradj Babich, Jonathan Alava, and Peter J. Clarke: *Towards Self-Testing in Autonomic Computing Systems.* International Symposium on Autonomous Decentralized System 2007: 51–58

Djuradj Babich, Kayan Chiu and Peter J. Clarke: *TaxTOOLJ – A Tool to Catalog Java Classes.* Software Engineering and Knowledge Engineering 2006: 375–380

David Crowther, Djuradj Babich and Peter J. Clarke: *A Class Abstraction Technique to Support the Analysis of Java Programs during Testing.* Southwest Educational Research Association 2005: 22–29

Peter J. Clarke, Djuradj Babich, Tariq M. King and B.M. Golam Kibria: *A Prediction Model for the Combination of Class Characteristics in Large OO Applications.* Florida International University - School of Computing and Information Sciences 2006