CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

Improving Software Fault Prediction using Novel Metrics based on Data Flow Volume and Coupling Complexity

by

Muhammad Rizwan

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Computing

Department of Computer Science

2022

Improving Software Fault Prediction using Novel Metrics based on Data Flow Volume and Coupling Complexity

By

Muhammad Rizwan

(DCS161001)

DEPARTMENT OF COMPUTER SCIENCE

CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY

ISLAMABAD

2022

To my parents, siblings, wife, and three angels, Ahmy, Faaty, and Aany.

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

Expressway, Kahuta Road, Zone-V, Islamabad
Phone:+92-51-111-555-666 Fax: +92-51-4486705
Email: info@cust.edu.pk Website: https://www.cust.edu.pk

CERTIFICATE OF APPROVAL

This is to certify that the research work presented in the thesis, entitled "**Improving Software Fault Prediction Using Novel Metrics Based on Data Flow Volume and Coupling Complexity**" was conducted under the supervision of **Dr. Aamer Nadeem**. No part of this thesis has been submitted anywhere else for any other degree. This thesis is submitted to the **Department of Computer Science, Capital University of Science and Technology** in partial fulfillment of the requirements for the degree of Doctor in Philosophy in the field of **Computer Science**. The open defence of the thesis was conducted on **February 23, 2022**.

Student Name : Muhammad Rizwan (DCS-161001) [illegible]

The Examining Committee unanimously agrees to award PhD degree in the mentioned field.

Examination Committee :

(a)	External Examiner 1:	Dr. Zohaib Zafar Iqbal Professor FAST-NUCES, Islamabad	[illegible]
(b)	External Examiner 2:	Dr. Onaiza Maqbool Associate Professor QAU, Islamabad	Onaiza
(c)	Internal Examiner :	Dr. Azhar Mahmood Associate Professor CUST, Islamabad	[illegible]

Supervisor Name :	Dr. Aamer Nadeem Professor CUST, Islamabad	[illegible]
Name of HoD :	Dr. Nayyer Masood Professor CUST, Islamabad	[illegible]
Name of Dean :	Dr. Muhammad Abdul Qadir Professor CUST, Islamabad	[illegible]

AUTHOR'S DECLARATION

I, **Muhammad Rizwan (Registration No. DCS-161001)**, hereby state that my PhD thesis entitled, '**Improving Software Fault Prediction Using Novel Metrics Based on Data Flow Volume and Coupling Complexity**' is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/ world.

At any time, if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my PhD Degree.

[illegible]

(**Muhammad Rizwan)**

Dated: February, 2022

Registration No : DCS-161001

PLAGIARISM UNDERTAKING

I solemnly declare that research work presented in the thesis titled "**Improving Software Fault Prediction Using Novel Metrics Based on Data Flow Volume and Coupling Complexity**" is solely my research work with no significant contribution from any other person. Small contribution/ help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero-tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/ cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of PhD Degree, the University reserves the right to withdraw/ revoke my PhD degree and that HEC and the University have the right to publish my name on the HEC/ University Website on which names of students are placed who submitted plagiarized thesis.

[illegible]

(Muhammad Rizwan)

Dated: February, 2022

Registration No : DCS161001

List of Publications

It is certified that following publication has been made out of the research work that has been carried out for this thesis:-

Journal publications:

1. **M. Rizwan**, A. Nadeem, and M. A. Sindhu,“ Vovel metrics—novel coupling metrics for improved software fault prediction”, *PeerJ Computer Science*, vol. 7, pp. e590, 2021.
2. **M. Rizwan**, A. Nadeem, and M. A. Sindhu,“Analyses of Classifier’s Performance Measures Used in Software Fault Prediction Studies”, *IEEE Access*, vol. 7, pp. 82764-82775, 2019.

Conference publications:

1. **M. Rizwan**, A. Nadeem, and M. A. Sindhu,“Theoretical Evaluation of Software Coupling Metrics”, 17th *International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 413-421, 2020.
2. **M. Rizwan**, A. Nadeem, and M. A. Sindhu,“Empirical Evaluation of Coupling Metrics in Software Fault Prediction”, 17th *International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 434-440, 2020.

Muhammad Rizwan
(DCS161001)

Acknowledgement

All praise is due to Allah Almighty, who has all knowledge and has the power to grant from his knowledge and peace, mercy and blessing upon his last and final messenger and upon good-doers.

I would never have been able to finish my work without the support from my respected parents and after that my very first teacher, my elder sister for her moral, physiological, and financial support and encouragement.

One of the most notable of Allah's blessings upon me was in the form of my supervisor. I would like to thank my supervisors, Prof. Dr. Aamer Nadeem for his guidance. I am also grateful to Dr. Muddassar Azam Sindhu, Assistant Professor Quaid-i-Azam University, whose reviews help me a lot to improve my work. I feel very lucky to be part of CSD research group members whose discussion and constructive criticism maintained an environment that was conducive for research. Moreover, without the recreational activities of our CSD research group, I may have gone insane over the last few years. I also thank the faculty members of Capital University of Science and Technology who gave me the resources and healthy education environment.

Very special thanks to my siblings for their motivation and encouragement. I would like to thank my beloved *Maana*. She was always there cheering me up and stood by me through the good times and bad.

I would like to express my gratitude to my friend and colleague Mr. Muhamad Yousuf Baig for his technical guidance. Apart from the mentioned above, many people helped me reach this point. May Allah grant all of them peace and bless them with prosperity.

Abstract

Well in time prediction of faulty modules is of key importance in software testing, which is generally done by software fault prediction (SFP) techniques. It lets testers focus more on faulty modules and prioritize test cases. It also assists in integration testing, hence significantly minimizes testing effort and improves testing quality.

The SFP community accepts the effectiveness of coupling between software modules in SFP. More specifically, coupling metrics which are purposed by Martin, Henry, and Chidamber are reported more useful. However, it is found that two important aspects of coupling, i.e., data flow volume and levels in software coupling have not been addressed so far. Keeping in view the same we proposed coupling metrics Vovel-in and Vovel-out, that incorporate these two aspects of coupling to improve the performance of SFP.

We performed experimentation by using five public datasets; Apache Lucene 2.4, Eclipse Equinox Framework 3.4, Eclipse JDT Core 3.4, Eclipse PDE UI 3.4.1, and Mylyn 3.1. These datasets provide class level information of numerous metrics along with the faults reported in each class. We selected five coupling metrics from the datasets due to their reported effectiveness in SFP. Finally, we extended the datasets by adding information of the proposed Vovel metrics from the projects' source code using JavaParser.

We first performed the univariate logistic regression to compute the significance of all the included coupling metrics, wherein all metrics were found significantly correlated with the fault. Later we performed the correlation analysis using Spearman correlation between all the coupling metrics in the datasets, to ensure the absence of duplicate information. It is observed that there is weak correlation exists between the metrics, yet not enough to be dropped. Finally, an experiment is conducted using multivariate logistic regression to analyze the performance achieved by including Vovel metrics. The significance of the result is ensured statistically using Wilcoxon test. The results of F-measure reflect significantly improved predictive performance of proposed metrics when used in combination with conventional class level coupling metrics.

In this thesis, we empirically evaluated the impact of coupling metrics, and more specifically, data flow volume coupling level in SFP. The results show that the inclusion of these factors significantly improves SFP.

Contents

List of Figures

List of Tables

Abbreviations

AM	Alghamdi metric
AST	Abstract Syntax Tree
AuC	Area under the receiver operator curve
C$\sim$	Total design complexity
Ca	Afferent couplings
CBO	Coupling between objects
CDM	Coupling dependency metric
Ce	Efferent couplings
CK	Chidamber and Kemerer metrics' suite
COF	Coupling factor
CPDP	Cross product defect prediction
CrtlC	Control coupling
CTA	Coupling through abstract data type
CTM	Coupling through message passing
DAC	Data abstraction coupling
DataC	Data coupling
DC	Design complexity metric
DCBO	Degree of coupling between objects
DM	Dhama metric
DL	Deep learning
DoP	Difference of opinion
E	Encapsulation
FM	Fenton and Melton metric
fp	Fault-prone

GIF	Global information flow
LIF	Local information flow
LOC	Lines of code
MDP	Metrics Data Program
ML	Machine learning
MPC	Message-passing coupling
NAS	Number of associations
NIHICP	Information flow-based non-Inheritance coupling
nfp	Not fault-prone
OCM	Operation coupling metric
OM	Offutt metric
OSGi	Open Service Gateway Initiative
OOP	Object Oriented Programming
RF	Random forest
RFC	Response for a class
SFP	Software fault prediction
SLR	Systematic literature review
UDP	Universal defect prediction

Chapter 1

Introduction

The pervasiveness of software in everyday life and in important decision-making tasks demands software to be more reliable than ever before. That amplifies the importance of software testing tasks that find the presence of faults in the software system. However, in order to discover all the residual faults, one should have to perform extensive testing. Such extensive testing is not feasible. It is reported that testing activity can potentially take 50% to 75% of the total development cost [1, 2]. Software failures cost the enterprise software market $61B annually [3]. This cost was raised to 1 trillion in 2016. It is considered as the most expensive activity in software development [4].

Fortunately, it is found that faults are having uneven distribution in the software system[5]. Few modules in a software product carry more faults than that of others. According to the study conducted by Gyimothy et al., fault are found in only 42% of the software products[6]. Likewise, another study suggested that about 70% of the faults are found by testing only 6% of the software modules[7]. Similar results are also reported by multiple studies conducted by Weyuker *et al.* [8–11].

Software fault prediction (SFP) is a process of detecting fault-prone (*fp*) module(s) or the number of expected faults in a software module. This is usually accomplished by employing an artifact from the same release of the software (Intra-release SFP) or from different releases of the software project (Inter-release SFP) or from the different software projects (Cross Project Defect Prediction) [12, 13]. The timely detection of faulty modules or the number of faults in any module is quite beneficial, especially, in critical and strategic software systems. It helps in reducing testing cost and improving the quality of the system [13]. Moreover, it can direct the testing efforts to focus more on the *fp* modules. Predicting the number of faults can be even more useful as it provides the criteria for *sufficient testing*. SFP assists in optimum resource utilization in the process of software testing[14].

There are three main ingredients in SFP [15], i.e, Metrics, Class label, and Model. Metrics is a measure that shows an aspect of software artifact (detail in Section 2.1.1). Class label is the required output, also known as independent variable. In SFP studies, class label can be numerical, that shows number of faults in a module, it can also be binary, like presence or absence of faults, or even ordinal like severity of faults. Finally, modeling maps the relationship between dependent to the independent variables. It is generally accomplished by ML algorithm, statistical algorithm, experts' opinion, etc. [16] (detail in Section 2.1.3). Among the existing metrics, modules' coupling is an important software design parameter. High coupling adds complexity and potentially leads to faults [17–22]. Moreover, coupling relations increase complexity, minimize encapsulation, reuse, and reduce understanding and maintainability[23]

1.1 Research Aims and Objectives

Modules' coupling is an important software design parameter and can potentially lead to software faults [17–22]. The research community reports the significant predictive performance of coupling in SFP. Our research aim and objectives start after appreciating the importance of coupling in SFP.

- There are many factors, which affect coupling complexity. For example, coupling between software modules by exchanging data has a different impact than that of coupling between software modules without the exchange of data. This exchange of data is referred to as volume. Likewise, coupling levels are another factor to influence the coupling between modules [24–27]. Hence, the coupling has *couple* of factors associated (see Section 3.3.5) that discriminate one coupling from another coupling. Some of these factors (like, volume of data flow, levels of coupling) have not been evaluated yet in SFP. Therefore, our objective is to evaluate the impact of these ignored factors in SFP. This objective can be achieved either by deriving new metric(s) that incorporate all these important factors or accumulating the existing metrics.

- SFP is generally done using ML models [16]. Selecting the right model and declaring the best out of multiple models are dependent on the performance measures. Our second objective is to find the most suitable performance measure out of the performance measures frequently used in SFP studies. Our analyses would help ML practitioners and researchers in SFP to select the most appropriate performance measure for the ML models' evaluation.

- Only seven coupling metrics have publicly available datasets [16]. These metrics do not provide any information about data flow volume and level of coupling. Therefore, our third objective is to develop defect datasets that contain information about data flow volume and levels of coupling also.

1.2 Research Questions

RQ-1 What are the different factors that contribute to coupling's complexity?
Objective: Since complexity potentially leads to faults. Our objective is to identify those factors that affect the coupling complexity and hence potentially leads to faults. These factors could be an important indicator in predicting *fp* modules.
Methodology: This research question would be answered by first finding different factors associated with modules' coupling. After that, we will examine the factors that are associated with coupling's complexity. The outcome of this methodology has been summarized in Table 3.5.

RQ-2 How different factors of coupling perform in predicting *fp* modules?
Objective: To assess the performance of coupling factors that can improve SFP.
Methodology: This research question would be answered by deriving new coupling metric(s) that incorporate important aspects of coupling. The newly derived coupling metric(s) would be first, searched in publicized datasets for availability. If not found, new datasets will be developed. In either case, complexity carrier factors of coupling metric(s) will be assessed by applying ML algorithm on the dataset(s). Chapter 6 has been dedicated to represents the performance of metrics that include volume of data flow and levels of coupling in predicting *fp* modules.
RQ-3 What is the most appropriate performance measure in SFP?
Objective: To identify the performance measures, that can be used to interpret the results and evaluate the performance of ML model.

Methodology: This question would be answered by first finding performance measures that are frequently used in SFP. After that, all the found performance measures are evaluated through some criteria to identify the best out of them. Section 5.3.1 provides the detail about the performance measures and finding the best of them in SFP.

1.3 Previous Work

Research community uses coupling and its different aspects in various dimensions including fault prediction [6, 19, 21, 23, 28–42, 42–59], design patterns [60], impact analysis [61], re-modularization [62], assessing software quality [39], maintenance cost [63], productivity [64], software vulnerabilities [65], reusability [66–80], changeability [81–96], reliability [97–105], and maintainability [42, 105–116]. However, keeping in view our scope we explore the coupling and its different aspects in the context of SFPs.
We performed survey in three different dimensions. The 1^{st} survey outputs the numerous coupling metrics proposed so far, followed by the 2^{nd} survey on SFP studies that use the metrics found in the 1^{st} survey. This survey will explore the studies on the empirical evaluation of the coupling metrics. Finally, 3^{rd} survey focuses the studies on coupling levels/principles to find the evaluation parameter for theoretical evaluation of the coupling metrics.
The research community reports the complexity due to high coupling. Numerous empirical studies also advocate the viability of coupling metrics in SFP [6, 19, 21, 23, 28–59, 117]. Coupling metrics in general, and Coupling between objects (CBO), Response for a Class (RFC), Fan-in, and Fan-out are specifically found useful in predicting software faults, irrespective of the dataset size, type of dependent variable. However, there are few metrics that have not been addressed by the SFP community. Mostly, coupling metrics are evaluated individually. Hence, the impact of multiple coupling metrics in SFP needs to be evaluated. SFP has been done using statistical techniques (i.e., Linear and Logistic regression), with the exception of few studies [6, 30] wherein Decision tree, Neural network, Support Vector Machines (SVM), Bayesian Learners. A wide variety of datasets are used by the research community which includes Public, Partially public, and Private. Likewise, all type of labels are used including Binary, Numerical and ordinal (severity and range of faults) are used as a label.

However, the coupling has many factors associated with it that add complexity [24–27]. After analyzing the existing coupling metrics proposed so far, it is found that all the coupling metrics do not incorporate the levels associated with the coupling. Therefore, there is a need of proposing a coupling metric that incorporates the coupling levels along with the volume of data flow between coupled modules. Since, these factors provide coverage of important coupling factors, thus they can potentially improve SFP process.
In addition to that, there are five features of coupling that are discouraged to be used in software system. These are; broad, remote, indirect, hidden, and rigid coupling[118]. The required new coupling metric is recommended for maximum coverage of coupling aspects.

1.4 Problem Statement

The coupling metrics used in the existing studies do not incorporate direction of control flow, coupling levels, and volume of information flow. Consequently, the exclusive impact of complexity associated with these aspects in SFP have not been discussed.

1.5 Research Contribution

- There are many factors, which contribute to coupling's complexity [24–27, 118]. Some of these aspects have not been evaluated yet in SFP. Two such aspects are the data flow volume and coupling levels. We propose two class-level coupling metrics that incorporate these two aspects. The proposed metrics are assessed in predicting software faults using 32 datasets. Finally, we reported the predictive ability of these two aspects of coupling.

- ML-based SFP models have been extensively used [16]. However, selecting the right model and declaring the best out of multiple models are dependent on the performance measures. We analyze 14 frequently used, non-graphic classifier's performance measures used in SFP studies. Our analyses would help ML practitioners and researchers in SFP to select the most appropriate performance measure for the models' evaluation.

- We perform an empirical evaluation of seven coupling metrics in SFP. Likewise, we perform theoretical evaluation of more than 37 coupling metrics against coverage of coupling levels and principles.

1.6 Research Methodology

The research methodology is a composite of the following four phases.
1st **Phase:** In this phase, we would identify the factors that contribute to complexity in the modules' coupling.
2nd **Phase:** In this phase, new metric would be equated by accumulating missing aspects of coupling.
3rd **Phase:** The usefulness of the proposed metrics is subject to be validated empirically. Availability of dataset is a base requirement for such validation. Therefore, this phase is dedicated for the development of dataset for the evaluation of the proposed metrics.
4th **Phase:** In this phase, the proposed metrics are evaluated for their impact on SFP. First, we perform the univariate logistic regression (ULR) to compute the significance of the coupling metric. The significant metrics are later assessed for the existence of association with other metrics using Spearman correlation coefficient. Later the least correlated metrics were used to build a multivariate logistic regression model.

1.7 Thesis Outline

The thesis outline is as follows:

Chapter 2 elaborates the SFP process and its three key ingredients: approaches, metrics, and datasets. After that performance evaluation techniques are discussed. Finally, software coupling, its principles, and its consequences are briefly disclosed.
Chapter 3 comprises a literature review in three different dimensions. The 1st section outputs the numerous coupling metrics proposed so far, followed by the 2nd section on SFP studies that use the metrics found in the 1st section. This section will explore the studies on the empirical evaluation of the coupling metrics. The 3rd section focuses the studies on coupling levels/principles to find the evaluation

parameter for theoretical evaluation of the coupling metrics. In the end, theoretical evaluation of coupling metrics is addressed w.r.t. the coupling levels and principles.

Chapter 4 describes the computation/derivation of our proposed metrics. The first half of the chapter elaborated the computation of data flow volume and coupling levels, while the second half is dedicated to illustrate the computing proposed metrics on hypothetical Java methods/classes.

Chapter 5 is dedicated to the selection and brief elaboration on materials and methods used for the empirical validation of the proposed metrics. This comprises dataset development, selection of performance measures, ML algorithm, and Statistical test.

Chapter 6 is dedicated to the reporting of the results computed using materials and methods discussed in the previous chapter. There comes a detailed discussion on the results along with the threat to validity.

Chapter 7 concludes the thesis and provides possible future directions for research.

Chapter 2

Background

Software cannot be free of errors, and every error can potentially lead to fault. Although, the density of fault can vary from industrial projects to normal applications [119, 119, 120], yet none is free from faults.x Testing is regarded as an activity of executing the software program and uncovering the faults as much as possible. The first known formal definition of software testing is defined by Glenford Myers as

> *Testing is the process of executing a program with the intent of finding errors*[121].

Since discovering all the faults in the software requires extensive testing, which is nearly impossible, therefore, software has to be deployed with those hidden faults. That is why software can never be guaranteed to be 100% reliable. An effective way to identify the fault is to determine the most fault-prone modules, which is the primary objective of software fault prediction.

2.1 Software Fault Prediction

SFP is the process of detecting *fp* module or the number of expected faults in a software module. This is usually, accomplished by employing an artifact from the same release of the software (Intra-release SFP) or from different releases of the software project (Inter-release SFP) or from the different software projects (Cross Project Defect Prediction) [12, 13]. The timely detection of faulty modules or the number of faults in any module is quite beneficial, especially, in critical and strategic software systems. It helps in reducing testing cost and improving the quality of the system [13]. Moreover, it can direct the testing team to focus more on the *fp* modules. Predicting the number of faults can be even more useful as it provides the criteria for *sufficient testing*. SFP helps in [14] the improvement of the software testing process and software quality. There are three main ingredients in SFP [15]

1. *Metrics* is a measure that shows an aspect of software artifact.
2. *Class label* is the required output, also known as independent variable. In SFP studies, class label can be numerical, that shows number of faults in a module, it can also be binary, like presence or absence of faults, or even ordinal like severity of faults.
3. *Model*. Modeling maps the relationship between dependent to the independent variables. It is generally accomplished by ML algorithm, statistical algorithm, experts' opinion. [16].

2.1.1 Metrics Used in SFP

SFP is performed using software metrics, which show the quantitative detail of software aspects. These metrics have two significantly distinct categories: process and product metrics [16].

2.1.1.1 Process Metrics

These metrics measure the development process that creates a body of software. A common example of a process metric is the length of time taken for the development of perticular software module. According to the study conducted by Catal et al. 79% of articles under study used process metrics for SFP [16].

2.1.1.2 Product Metrics

These metrics describe the characteristics of the product such as size, complexity, design features, performance, and quality level. Within the product, metrics can vary in levels. According to the study conducted by Catal et al. 60% of articles under study used method-level metrics [16](see Figure 2.1). There is another dimension of product metrics, there could be design metrics, code metrics, volume metrics, complexity metrics. This document exclusively considers only the product

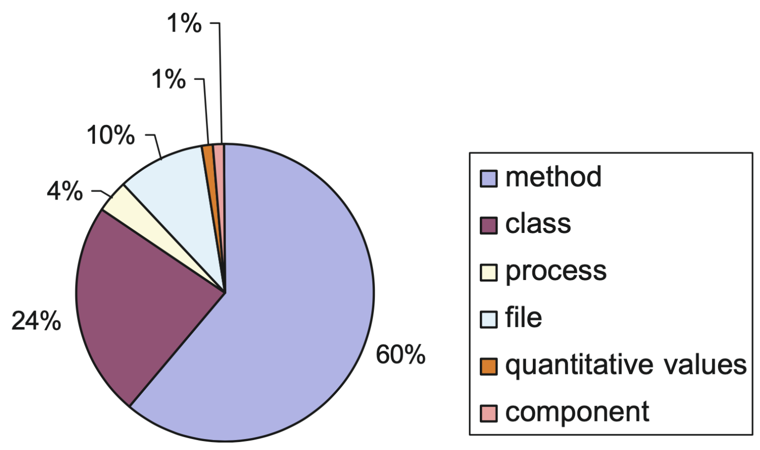


Figure 2.1: Distribution of metrics

metric, assuming uniformity in processes within a product. According to the studies, [15, 16, 22, 122–126] the most frequently used in structural paradigm are McCabe [127], Halstead [128] and LOC. In object oriented paradigm, metrics are generally proposed in suites like Morris metric suite [129], CK metric suite [130], Oman coupling metric suite [131], Chen and Lu metric suite [132], Li and Henry metric suite [106], Lee *et al.* metric suite [133], Abreu et al. metric suite [134], Martin metric suite [135], Lorenz and Kidd metric suite [136], Briand *et al.* metric suite [61], and Li metric suite [137]. However, CK metric suite is the most frequently used [15, 16, 22, 122–126]. Besides this, inheritance metrics are reported as the least useful or insignificant metrics. More specifically, Depth of Inheritance Tree (DIT) and Number of childern (NOC) of the Chidamber and Kemerer inheritance metrics are extensively evaluated and found insignificant. Likewise, Lacks of Cohesion Metrics (LCOM), LCOM2, LCOM3 are also found useless in predicting software faults [138].

2.1.2 Datasets Used in SFP

Dataset is a collection of multiple records/instances that comprises one or more metrics and a class label. Numerous datasets have been used in fault prediction studies. Based upon their availability, datasets can be classified into three classes:

public, partial public, private[16, 22].

Publicly available datasets comprise the labeled datasets having a value of some metrics against each instance. The major datasets used are as follows:

1. NASA [139] datasets consists of 13 software projects in PROMISE version and 14 software projects in metrics data program (MDP) version, with considerable difference in the number of features set also [140]. Besides having the quality problems [140], it is the most used dataset by the SFP community [138].

2. SoftLab [141] dataset contains five projects, i.e., Autoregressive-1 (AR1), AR2, AR3, AR4, and AR5, which are embedded controller software for white goods. All these projects are written in C.

3. AEEEM [142] is collected by D'Ambros *et al.*. It contains class-level datasets of five software systems; *Apache Lucene 2.4, Eclipse Equinox Framework 3.4, Eclipse JDT Core 3.4, Eclipse PDE UI 3.4.1, and Mylyn.* These datasets have 61 metrics, which include code metrics, process metrics, and churn metrics.

4. ReLink [143] is a file-level dataset of three projects; Apache, Safe, and ZXing having 194, 56, and 399 instances, respectively. Each dataset contains 26 features, including complexity and count metrics.

5. Jureczko and Madeyski[144] collected 92 versions of 38 different software development projects. These projects are open source, proprietary and academic software projects. Each of these datasets contains 20 metrics, including McCabe's cyclomatic metrics, CK metrics, and other object oriented metrics. Few of these datasets are Tomcat, Ant, Camel, Ckjm, Forrest, Ivy, JEdit, Log4j, Lucene, PBeans, Poi, Prop, Synapse, velocity, Xalan, Xerces, etc.

6. ECLIPSE1 [145] contains file and package level datasets of three versions of Eclipse, i.e., 2.0, 2.1, and 3.0. These datasets contain code metrics on the file level and 40 metrics on the package level.

In *Partial public* datasets, source code of the software system and fault data is available, from where the metrics can be parsed, like Eclipse [145] and Mozilla.

Few studies which use these datasets are [29, 32, 36, 46, 52, 54]. However the most critical issues with partial public are many-to-many relation between bug-file links, duplicated bug-file links and the issue of untraceable bugs [146].
In *Private* datasets, projects' code and defect information are not public. These datasets are mostly proprietary used by few studies [31, 34, 37–39, 45, 58, 59, 147]. This type also includes projects developed by the university students.
Catal and Diri [16] conduct a systematic literature review (SLR) from the years 1990 to 2007. They conclude that 69% of papers have private datasets published before 2005, which becomes 48% after 2005. SLR conducted by Radjenvic *et al.* [22] covers the studies published between 1991 and 2011 declares that 62% of the studies used private, 22% partial public datasets. Ronald [148] reports that from 1996 to 2012, 57% of the studies used publicly available datasets. A survey conducted by R. Malhotra concludes that 23% of the paper published from 1991 to 2013 use private dataset, while 77% of papers use public dataset [123]. This collectively implies that researchers are more inclined towards public datasets. The reason is the least availability of fault information.

2.1.3 Approaches Used in SFP

Five different approaches are being used in SFP;

1. Machine learning
2. Deep learning [149, 150]
3. Statistical modeling
4. Experts' opinion
5. Similarity based [151]
6. Association rule mining [152–155]

In the SFP literature, most of the articles used ML-based techniques. [16]. The usage frequency of ML algorithms from 1991 to 2013 shows the domination of decision tree and Bayesian[7, 123]. Lately, a study conducted by Mahesh *et al.* supports the frequent use of ML for fault prediction[156]. However, as far as performance is concerned, generally, models perform best where the selected technique successfully maps the relationship between dependent to independent variables. A survey is conducted that includes studies published from 1991 to 2013. The objective was to assess the performance of ML algorithm in terms of AuC. It has been reported that the RF performed the best, followed by the Multilayer perceptron (MLP), Naive Bayes (NB), and Bayesian Network (BN) models [123]. The most recent survey is conducted with coverage from 1995 to 2018 [138]. The authors reported that the most used learning method is Decision Tree. The second most used technique is built on Bayes theorem. While, Regression, Discriminant analysis, and Threshold-based classification are the third most used technique for SFP. Apart from that, RF is generally appreciated by the SFP research community [35, 49, 50, 157–163].

2.1.4 Performance Evaluation in ML-based SFP Techniques

There are numerous performance measures used in the evaluation of the SFP machine learning models. Some commonly used performance measures are Accuracy, Recall, Precision, F-measure, and AuC. However, the most used performance measure is Recall followed by Precision and AuC [138].
We analyzes 14 frequently used, non-graphic classifier's performance measures used in SFP studies. The objective of the study is to help ML practitioners and researchers in SFP to select the most appropriate performance measure for the models' evaluation. They evaluate the performance measures for resilience against producing invalid values through their proposed plausibility criterion. After that, consistency and discriminant analyses are performed to find the best out of the 14 performance measures. Finally, they conclude that the F-measure is the best candidate to evaluate the SFP models.
Malhotra [123] report that amongst the studies published from 1991 to 2013, F-measure is amongst the top five most-used performance amongst 21 collected performance measures. Likewise, SLR of the studies from 1998 to 2018 also reports the same result [138]. Keeping in view these findings we select F-measure as a performance evaluation measure.
F-measure [164] is the harmonic mean of Precision and Recall. It tells how precise a classifier is and how robust it is. Mathematically it is written as.

$$F\text{-}measure = \frac{(\beta^2 + 1) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{2.1}$$

Whereas the value of β can be 0.5, 1, and 2 for F0.5, F1, and F2 measures, respectively, where F1-measure is being the most used variant.
Apart from that, some of the studies use statistical tests for the evaluation of their results. In this context, Wilcoxon test is the most used test followed by the Friedman and Analysis of Variance (ANOVA) test [138].

2.2 Software Coupling

Software coupling is defined as,

"The measure of the strength of association established by a connection from one component to another" [165, 166]

In software engineering, the coupling can be in different ways, like:

1. OO paradigm vs Aspect-oriented paradigm [167].
2. OO paradigm vs structural paradigm
3. Static coupling vs dynamic coupling [168, 169]
4. Syntactic coupling vs conceptual/semantic coupling [169, 170]
5. Coupling through inheritance vs coupling without inheritance

In this document, we are concerned with syntactic, static coupling in object oriented and structural paradigm without inheritance. More precisely, the coupling is considered in SFP, instead of reusability, maintainability, etc.

2.2.1 Coupling Principles and Consequences of High Coupling

There are five principles of coupling [118]. Which are flexible, obvious, local, direct, and narrow connections between software modules are desirable. In contrast to that rigid, hidden, global, indirect, and broad connections should be avoided. Coupling does not hold transitive property [130].

High coupling is undesirable, as it leads to complexity and consequent faults. The reason is that highly coupled modules are difficult to reuse, modify or test without understanding all the modules that the very module is coupled to. If an error occurs in a highly coupled module then the probability of an error in other modules increases. That is why highly coupled modules can be more *fp* [171, 172]. A comprehensible level of coupling is always desirable, however, when there exists complexity in coupling in terms of levels, and volume of information flow there comes incomprehensibility. Consequently, the programmer finds it difficult to program by keeping all parameters in consideration and thus causes him to commit a bug in a code [173]. Offutt [25] describes that high coupling increases the:

- probability to propagate faults from one module to another.
- effect of change impact from one module to another.
- time to understand the single component.

Coupling is closely related to cognition. This is an empirically proven fact[21, 38, 40, 41], which can be pictorially shown as in the Figure 2.2

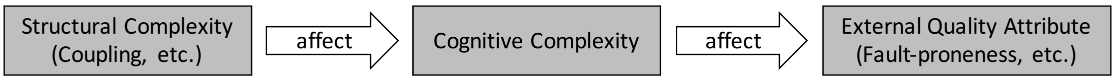

Structural Complexity (Coupling, etc.)
affect
Cognitive Complexity
affect
External Quality Attribute (Fault-proneness, etc.)

FIGURE 2.2: Relationship of structural complexity of the software with software faults

Inheritance is another software design aspect, and it is a unique type of coupling. Since, it enables re-usability [130] and has a very weak association with software faults[23, 174]. Keeping in view these reasons, we exclude it from our thesis.

Chapter 3

Literature Review

Research community uses coupling and its different aspects in various dimensions including fault prediction [6, 19, 21, 23, 28–42, 42–59], design patterns [60], impact analysis [61], re-modularization [62], assessing software quality [39], maintenance cost [63], productivity [64], software vulnerabilities [65], reusability [66–80], changeability [81–96], reliability [97–105], and maintainability [42, 105–116]. However, keeping in view our scope we explore the coupling and its different aspects in the context of SFPs. This chapter comprises a survey in three different dimensions. The 1st survey outputs the numerous coupling metrics proposed so far (Section 3.1), followed by the 2nd survey on SFP studies that use the metrics found in the 1st survey (Section 3.2). This survey will explore the studies on the empirical evaluation of the coupling metrics. Finally, 3rd survey focuses the studies on coupling levels/principles (Section 3.3) to find the evaluation parameter for theoretical evaluation of the coupling metrics. Figure 3.1 illustrates the orientation of this chapter.

3.1 Studies on the Development of Coupling Metrics

In this section, we briefly discuss the coupling metrics in chronological order. Most of the coupling metrics discussed here are part of some metric suite (see Appendix A). The following section precisely defines the inclusion and exclusion criteria

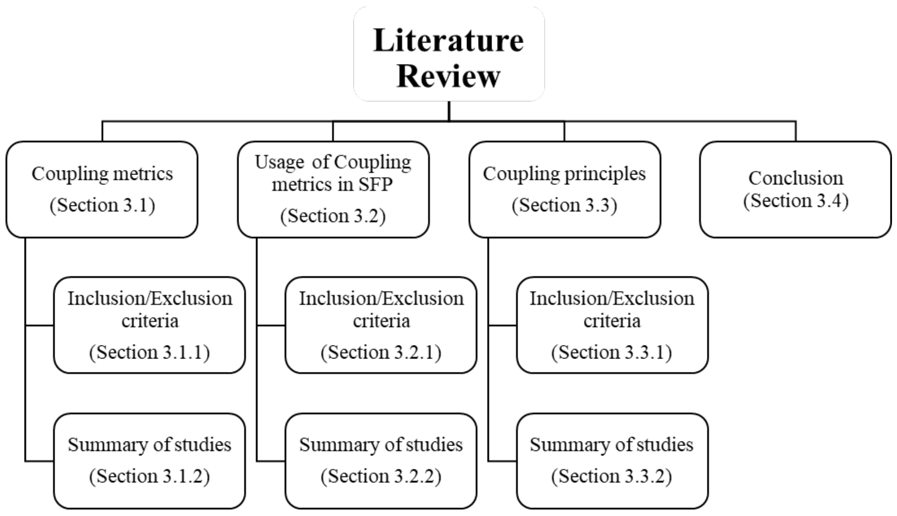

Literature Review
Coupling metrics (Section 3.1)
Usage of Coupling metrics in SFP (Section 3.2)
Coupling principles (Section 3.3)
Conclusion (Section 3.4)
Inclusion/Exclusion criteria (Section 3.1.1)
Inclusion/Exclusion criteria (Section 3.2.1)
Inclusion/Exclusion criteria (Section 3.3.1)
Summary of studies (Section 3.1.2)
Summary of studies (Section 3.2.2)
Summary of studies (Section 3.3.2)

FIGURE 3.1: Orientation of Literature review

followed by the section that summarizes the included studies.
We do not conduct a systematic literature survey, however, we tried to include maximum studies through snowball sampling [175] and random query generation on digital libraries. Fortunately, our list covers all the coupling metrics discussed in various Systematic literature surveys (SLR) [16, 22, 148, 176–184].

3.1.1 Inclusion Criteria

In this literature review we include following type of studies

1. Studies wherein the coupling metric has been proposed with focus on syntactic aspects
2. Studies wherein the coupling metric has been proposed with focus on static aspects
3. Studies wherein the coupling metric has been proposed for object oriented paradigm or structural paradigm

3.1.2 Exclusion Criteria

In this review we exclude the studies having following properties

1. Studies wherein the coupling metric has been proposed with focus on conceptual/semantic aspects
2. Studies wherein the coupling metric has been proposed with focus on dynamic aspects of coupling
3. Studies wherein the coupling metric has been proposed that addresses any of the aspects of inheritance like Coupling based on Strength Specification Metric (CSSM) [185, 186]
4. Studies wherein the coupling metric has been proposed *informally*, yet are used by some studies like Access To Foreign Data (ATFD), Foreign Data Providers (FDP), Locality of Attribute Accesses (LAA), etc [187]

3.1.3 Summary of the Coupling Metrics

3.1.3.1 Henry and Kafura Coupling Metrics

The first coupling metrics by Henry and Kafura [17, 18]. The authors proposed an four different types of information flow complexity measure. In this context, relationships between software modules/programs are either fan-in's or fan-out's. Likewise local and global information flow coverage are equated for relationship between variables and program modules. That makes a total of four coupling metrics. Two metrics are represent coupling software modules, i.e., Fan-in and Fan-out. Whereas two coupling metrics represent coupling between variables and software modules, i.e, Global information flow and Local information flow, shortly named as GIF and LIF, respectively.

3.1.3.2 Total Design Complexity

Total design complexity ($C\sim$) [188] calculates the overall system's complexity by adding of inter-module and intra-module complexity. Mathematically, $C\sim$ of a method A is;

$$C \sim (x) = \frac{\sum {f_{\mathrm{x}}}^2}{n} + \frac{\sum \frac{v_{\mathrm{x}}}{f_{\mathrm{x}}+1}}{m} \tag{3.1}$$

Where, f_x is the frequency of method being called by other modules (i.e Fan-out of module x), n represents the total number of modules in the software project, v is the total number of input/output variables variable in module x and m is the number of new modules in the system. Desirable value of C$\sim$ metric $\leq$ 26 [189].

3.1.3.3 Degree of Coupling Between Objects

Degree of Coupling Between Objects (DCBO) [129] computes the number of uses of one object to the elements of another object. It can be found by simply drawing a number of arcs between objects in a network diagram.

3.1.3.4 Design Complexity

Design complexity metric (DC) [190] is the cyclomatic complexity [127] of the reduced control-flow graph. The reduction is done on some rules. With no function call, design complexity would be 1. However, for every Fan-out in a sequence or branch, DC would increase by 1. DC is similar to Fan-out, but it does not consider updating any global data structure.

3.1.3.5 Fenton and Melton Metric

Fenton and Melton metric(FM) is an ordinal metric of coupling between any two modules [191]. The authors have proposed the metric as a measure of coupling between two components x and y :

$$M(x, y) = i + \frac{n}{n+1} \tag{3.2}$$

where, n =numberofinterconnectionsbetween x and y, and i = level of highest (worst) coupling type found between x and y.

3.1.3.6 Chidamber and Kemerer Metrics Suite

Chidamber and Kemerer present suite which comprise two coupling metrics [130], i.e., Coupling between objects (CBO) and Response for a class (RFC).
CBO is the total number of classes that are coupled to a particular class. CBO could occur if the method(s) of a class invoke(s) member (data member or member function) of another class. If class A invokes member(s) of class B and C then CBO of A will be 2. Multiple accesses of a particular class's member would be counted as a separate occurrence [192]. CBO ≤ 14 is acceptable [193]. CBO came with two versions, the earlier version does not include inheritance-based coupling [130] while latter does [194]. In this study, an earlier version of CBO is considered in evaluation, literature review and later in the experimentation.
RFC of any class C is a union of two sets:

1. Set of all methods of C
2. Set of all methods of classes other than C which is called by method(s) of C

Multiple calls to the same method would be counted only once.

3.1.3.7 Oman Coupling Metrics

Oman and Hagemiester propose four coupling metrics [131]. The original manuscript is not available, however, the details about the coupling metrics are discussed in later article by the same authors [109].
The mathematical equation of these metrics are as follows

$$ControlCoupling(CrtlC) = Fan - out^2 \tag{3.3}$$

$$DataCoupling(DataC) = \frac{Total\,Global\,Structures + Modules'\,Parameters}{\dfrac{No.\,of\,Data\,Structures\,in\,module}{No.\,of\,Modules}} \tag{3.4}$$

$$Encapsulation(E) = \frac{1-(DataCoupling \times No.\,of\,Modules)}{Fan-in} \tag{3.5}$$

$$Module\,Resue(MR) = 1 - \frac{No.\,of\,Modules}{Fan-in} \tag{3.6}$$

$$System\,Coupling(SC) = \frac{No.\,of\,Global\,structures + No.\,of\,parameters}{No.\,of\,Data\,structures} \tag{3.7}$$

3.1.3.8 Chen and Lu Coupling Metrics

There are two coupling metrics in this suite [132]. Operation coupling metric (OCM) and Class coupling metrics (CCM).
OCM measures the coupling between the operations of two classes. Mathematically,

$$\begin{aligned} OCM = {} & No\,of\,Oper.\,accessotherclasses \\ & + No\,of\,Oper.\,accessedbyotherclasses \\ & + No\,of\,Oper.\,cooperated \end{aligned}$$

When class directly/indirectly accesses members of other class, while that called class is accessing members of that very class, cooperated coupling is established.

$$\begin{aligned} CCM = {} & No.\,of\,Classeswhichaccessotherclasses \\ & + No.\,of\,Classes\,being\,accessed \\ & + No.\,of\,cooperated\,Classes \end{aligned}$$

CCM and OCM[132] are having subtle differences. OCM is more generalized in its coverage. Since it covers data members and member function both. Whereas, CCM is the operation of accessing other classes.

3.1.3.9 Offutt Metric

Offutt metric (OM) [26] precisely define the coupling levels that can be computed through an algorithm. Moreover, the authors incorporated the notion of coupling direction also. The coupling levels used in OM are shown in Table 3.4.

3.1.3.10 Message Passing Coupling and Data Abstraction Coupling

Li and Henry propose two coupling metrics [106]; Message-passing coupling (MPC) and Data abstraction coupling (DAC).
MPC computes the coupling between classes through message passing. Hence, it is computed at the class level instead of the object level. It shows the number of messages sent out of the class, whereas DAC is the number of abstract data types in a class.
The definitions of both of these metrics are descriptive in the original study, however, another article [195] defined them as;
MPC is “the number of method invocations in the class of interest” and DAC is “the number of attributes in a class of interest whose types are of other classes.
Later, Wei Li [137] proposes six metrics wherein two are related to coupling, which are very similar to his earlier study [106]; Coupling Through Abstract Data Type (CTA), and Coupling Through Message Passing (CTM). The CTA metric is the total number of classes that are used as abstract data types in the data- attribute declaration of a class. The CTM metric measures the number of different messages sent out from a class to other classes.
Since both of these metrics have no significant difference in these two studies, we consider them as single versions. Therefore, in the rest of this document, these two versions are referred to by their prior versions, i.e., DAC and MPC.

3.1.3.11 Coupling Factor

Coupling Factor (CLF) takes into account the main components of OOP including inheritance, encapsulation, information hiding or polymorphism with the belief to be responsible for the increase in software quality and development productivity [196]. Mathematically it is;

$$CLF = \frac{TCC}{TC} \tag{3.8}$$

Whereas where TC is the total number of classes in the system under consideration and Class Clusters are represented by TCC.

3.1.3.12 Lee *et al.* Coupling Metrics

Lee *et al.* propose a metric suite wherein two metrics are related to coupling [133]. These two metrics are Information flow-based non-Inheritance coupling (NIHICP) and Information flow-based coupling (ICP). NIHICP [133] is the number of methods that are invoked by other classes. These methods are weighted by the number of parameters of the invoked method.

3.1.3.13 Dhama Metric

Dhama metric (DM) [197] incorporates four aspects of coupling; global coupling, data coupling, control coupling, and external in his proposed metric. In DM, low coupling is good which is shown by high number and vice versa. So, coupling C is inversely proportional to coupling factors. $K = 1$ is taken as proportionality constant. Mathematically,

$$C = \frac{1}{SUM(p_iC(i), p_oC(o), p_gC(g), D(i), D(o), D(g), Fan-out, Fan-in)} \tag{3.9}$$

Whereas total number of input, output, and global parameters which are used for data coupling are shown by D(i), D(o), and D(g), respectively. Likewise, total number of input, output, and global parameters which are used for control coupling are shown by, Ctr(i), Ctr(o), and Ctr(g) respectively.

$p_i, p_o, and p_g$ are given value as 2 as heuristic estimate. The authors came up with additional levels of coupling by minor augmentation of the existing coupling metrics purposed earlier [24–27].

3.1.3.14 Afferent and Efferent Couplings

Martin propose three coupling metrics [135]: Afferent Couplings (Ca) is the number of classes outside this category that depend upon classes within this category. Efferent Couplings (Ce) is the number of classes inside this category that depend upon classes outside this categories. Third metric is Instability (I), which is mathematically written as,

$$I = \frac{Ce}{Ca + Ce} \tag{3.10}$$

The value of Instability varies from zero to one. The lower value represents a maximally stable category, while a higher value represents a maximally unstable category.

3.1.3.15 Coupling Dependency Metric

Coupling dependency metric (CDM) considers three dependencies in the coupling; Referential (R), structural (S), and integrity (D) dependency [198]. The metric is equally applicable in the structural and OO paradigm. The metric incorporates levels of coupling also.

3.1.3.16 Ordinal Scale Module Coupling

Ordinal scale module coupling (OSMC) [43] takes into account five levels of couplings. These are Content, Common, Control, Stamp, and Data. The metric is discussed descriptively without any mathematical form. However, the authors claim a strong correlation between OSMC and failures.

3.1.3.17 Briand's Coupling Metrics

Briand *et al.* [199] propose a number of metrics which count the number of class-attribute (CA), class-method (CM) and method-method (MM) interactions for each class. CA occurs when one class has an attribute of another class. CM interaction occurs when one class has a method and having a parameter of another class type. MM interaction type occurs when a method of one class calls a method of another class. By the inclusion of Friend, Inverse friend, Afferent, and Efferent coupling, a suite of metrics is proposed. Cartesian product of these concepts makes in total 18 metrics in number (and ignoring infeasible combinations). However, 6 metrics are related to inheritance so they are ignored. Thus, remaining relevant metrics are FCAEC, FCMEC, FMMEC, IFCAIC, IFCMIC, IFMMIC, OCAEC, OCAIC, OCMIC, OCMEC, OMMEC, OMMIC.

3.1.3.18 Number of Associations

Number of associations (NAS) [192] is the number of association lines spread out from a class on an Object Model diagrams [200]. There is a subtle difference between NAS and CBO. The NAS considers only the unique calls, whereas CBO considers each call as a separate occurrence. Moreover, NAS is similar to another metric, i.e, DCBO [129].

3.1.3.19 ABC Metric

In ABC metric [201], A is for assignment, usually done by transferring data into a variable, B is for Branch, means when control moves out of the module's boundary, C is for condition, which is generally a Boolean check. So, higher the value of B (which is there in case of coupling), higher the value of ABC metric. Mathematically ABC of any method A is;

$$ABC(A) = \sqrt{A + B + C} \tag{3.11}$$

3.1.3.20 Direct Class Coupling

Direct class coupling (DCC) is a part of QMOOD metric suite [202]. It is a different number of classes that a class is directly related to by attribute declaration or message passing.

3.1.3.21 Cognitive Functional Size Metric

This metric takes into account the functional complexity in terms of cognition[203]. CFS takes into account the weight assigned to basic control structure in a module. Apart from this every functional call has been assigned a weight of 2. The value of the metric highers by high frequency of function calls.

3.1.3.22 Nagappan and Thirumalesh Coupling Metrics

Nagappan and Thirumalesh propose multiple coupling metrics [204] and make them patent.

3.1.3.23 Weighted Transitive Coupling

Weighted transitive coupling (WTrnCpl) takes into call graph in to consideration[66]. Its mathematically form is as follows;

$$WTrnCpl = \frac{\sum_{x,y=1}^{n} Cpl(x,y)}{n^2 - n} \tag{3.12}$$

Where n represents the total count of modules in the software system under consideration. Whereas, $Cpl(x, y)$ utilizes the relationship between two modules, i.e., x and y.

3.1.3.24 Alghamdi Metric

Alghamdi metric (AM) [205] used to compute coupling metric by first populating a matrix named Description matrix, which is a Cartesian mapping of program components. After that coupling metric is computed by taking into account the values placed in the cell.

3.2 Studies on the Evaluation of Coupling Metrics

In this section, we discuss the studies which evaluate coupling metrics in SFP. The objective of this section is to assess the importance of coupling metrics empirically by the research community. The following subsection precisely defines the inclusion and exclusion criteria followed by the section that summarizes the included studies in chronological order.

3.2.1 Inclusion Criteria

In the section, we include the studies having all of the following properties

1. Studies that report the exclusive impact of coupling metrics discussed in Section 3.1.3
2. Studies from software fault prediction domain
3. Studies wherein machine learning is used for software fault prediction

3.2.2 Exclusion Criteria

In the section, we exclude the studies having any of the following properties

1. Studies that report the metrics other than the coupling metrics discussed in Section 3.1.3
2. The studies that address the coupling metrics other than SFP are not included.

3.2.3 Summary of the Included Studies

Troy and Zweben [19] use coupling measures that are related to any aspect of coupling like the number of interconnections per module (volume), the number of modules accessing a common interconnection (frequency of common coupling), unique common interconnections (common coupling), boxes accessing control interconnections (control coupling). The authors test the hypotheses for the usefulness of the coupling indicators. Data is taken from 73 designs and their corresponding source code. The result of linear regression modeling indicates that module coupling is an important factor in determining the quality of the software. **Kitchenham** [206] assesses multiple design metrics that are based on Henry and Kafura's information flow metrics (i.e. Fan-in and Fan-out). A communications system is taken as a case study. The objective is to evaluate the ability of selected metrics to identify change-prone, error-prone, and complex programs. Based upon visual scatter plots, it is reported that the Fan-out has a strong association with software fault, whereas Fan-in is relatively weak in this trait.

Bisili *et al.* [45] evaluate the performance of CBO and RFC in predicting the fault-prone modules. The authors take a total of eight medium-sized software systems. It is concluded that the included coupling metrics significantly improves the predictive accuracy.

Binkley *et al.* [44] take corrective maintenance data of a 82,000-line code developed in C++. The software is a patient collaborative care system. In addition to the report of insignificance of inheritance metrics, the authors reported the effectiveness interclass and intraclass metrics are in fault-prone prediction. The included coupling metrics are CBO, NCC, NSSR, CDM, Fan-in, Fan-out. The correlation is performed using Spearman correlation.

Binkley and Schach [43] perform a study to determine the usefulness of design-based software product metrics in predicting the number of run-time faults that will be encountered during execution. The included coupling metrics are Fan-in, Fan-out, OSMC, and CDM. It is deduced that the most effective metric for predicting failures is related to coupling. In other words, the metrics which correlated most highly with failure were those which measured some form of module coupling.

Binkley and Schach [117] investigate the usefulness of CDM, OSMC, Fan-in, and Fan-out in predicting run-time failures using Spearman correlation. OASIS is taken as a case study that is developed in COBOL. It is reported that the most accurate predictor of run-time failures is the amount of interdependency between modules, which is computed by the selected coupling metrics.

Binkley and Schach [42] employ four case studies: run-time failure data for a COBOL registration system; maintenance data for a C text-processing utility; maintenance data for a C++ patient collaborative care system; and maintenance data for a Java electronic file transfer facility. It is found that the most undesirable property in terms of software maintenance is the level of coupling between modules. In contrast to that, the case studies with lower coupled modules undergo less maintenance. More specifically, That is, coupling metrics in general, and CDM in particular, are excellent predictors of maintenance measures.

Briand *et al.* [41] investigate the usefulness of existing coupling metrics in identifying the probability of fault detection. Both import and export couplings are used as independent variables. Medium-sized eight different software systems developed by students are used for the evaluation. Fault data is taken from the independent testing team. Included coupling metrics are Henry and Kafura coupling metrics and Briands coupling metrics. The result of regression coefficients shows that all the coupling metrics are a good predictor of software faults except OCMIC.

Harrison *et al.* [192] hypothesized that coupling is associated with a number of errors and understandability. The authors analyzed two coupling metrics CBO and the NAS. A strong association is found between these two metrics. Both the metrics are available early on in system design. The experiment is performed on five object-oriented systems which are developed in C++. The results of the Spearman correlation show the existence of partial association included coupling metrics with fault density. More specifically, NAS is more associated with the density of faults than that of CBO.

Eman *et al.* [40] assess the significance of CK and Briand's coupling metrics in predicting fault-prone modules. The independent variables are measured using an open source software tool and configuration management is used for labeling the faulty classes. Logistic regression is used for model building. The performance evaluation is done by using of R^2 and coefficients. The outcome of the experiment infers the positive association of included metrics related to coupling with fault. More specifically, CBO, OCMEC, OCAEC, and OMMEC are found a prudent choice for fault prediction.

Mei *et al.* [59] evaluate CK metric suite using univariate logistic regression. They made three classes of faults; Object-oriented faults, Object management faults, and Traditional faults. They report the usefulness of RFC. Moreover, the authors propose few metrics and found them useful also.

Briand *et al.* [39] explore the association of import/export coupling measures and the probability of fault detection. The eight systems used for this study were developed in C++ by students over the course of four months. The systems consist of 180 classes. Coupling metrics are parsed using M-System and fault data is collected during the testing phase which is conducted by an independent testing team. They conclude that coupling measures, with good variance, are significantly useful in predicting software faults. The result of univariate logistic regression is that all import and export couplings are useful in SFP except OCAEC.

Briand *et al.* [38] hypothesized the association of numerous coupling metrics with software faults. The independent variables comprise the CK and Brainds coupling, which are parsed using a tool developed at the Frauhofer IESE. The object-oriented based industrial software system written in C++ is used as a case study. The fault information is computed using the change report form. The results of univariate logistic regression report the strong association of all included coupling metrics with software faults.

Eman *et al.* [58] apply logistic regression and Pearson correlation on the telecommunications framework written in C++. They evaluate the association of CBO, RFC, and Briand's metric suite with software faults. They report that

CBO and RFC are both associated with faults, whereas RFC's association gets weaker when size is controlled.

Eman and Melo [21] evaluate Briand's coupling metrics in SFP. A software system developed in Java is used as a case study. The fault and fault-free instances are determined by skimming the failure reports. The experimental methodology is cross-version defect prediction. One version of the software is used for building a logistic regression model. The built model is used for the fault module identification in the subsequent version of the same project. The result is reported in the form of Regression coefficients. It is deduced that there exists a strong association between coupling metrics and faults. The conclusion is the same as that reported by El-Eman [40].

Ping *et al.* [37] empirically validate a set of object-oriented metrics in terms of their effectiveness in predicting fault-prone software modules. The authors use a total of ten metrics including, size, coupling, cohesion, and inheritance. Out of the total of eight hypotheses, three hypotheses exclusively address the significance of coupling metrics (CBO, RFC, Fan-in). Client-side of a large network service management system developed in Java is used as a case study. The evaluation of results is carried out using regression analysis and discriminant analysis. The result of R^2 advocates the usefulness of CBO and RFC. While mild effectiveness is reported in favor of Fan-in.

Subramanyam and Krishnan [207] investigate the performance of CBO (and some non-coupling metrics) in SFP. The study uses an e-commerce application suite developed in C++ and Java, wherein total classes are 706. Metrics are computed from the design document and source code. Defect data is collected from customer acceptance testing and defect resolution logs, which are later validated by the concerned development team. They examined the effect of the size along with the CBO values on the faults by employing multivariate regression. Besides validating the usefulness of metrics, they compared the applicability of the metrics in different languages; thus, they test the hypotheses for C++ and Java classes separately. The results show the usefulness of CBO in C++ projects.

Gyimothy *et al.* [6] investigate eight metrics, including CK metrics, LCOMN, and LOC. Every metric is evaluated independently. The authors process the Bugzilla database with bugs associated with each class in the source code. Both classification and regression are performed on the Mozilla software project using logistic and linear regression respectively. R^2 is used as a performance evaluation parameter. The results infer the effectiveness of CBO for having its large value of R^2. While mild usefulness of RFC is observed.
Finally, another experiment is performed using a decision tree and multilayer perceptron on the binary and ordinal fault data. The performance evaluation is conducted using three performance measures. However, outcome of the experiment supports the results reported when regression is performed.

Abubakar *et al.* [208] aimed to assess the effect of cohesion metrics on software faults. They perform stepwise linear regression on KC1 dataset wherein PPD, ATPD, CBO, DIT, LCOM, NOC, RFC, WMPC, and DOC are used as independent variables. Though the objective was the evaluation of cohesion metric LCOM, a significant correlation of CBO and WMPC with software faults is reported. However, Fan-in is declared insignificant.

Janes *et al.* [147] perform three regression techniques on five real-time telecommunication system. The objective was to assess the performance of CK metric suite in fault prediction. They report the statistically significant performance of RFC in all the projects, while CBO found it useful on some of the analyzed projects.

Shatnawi *et al.* [36] assess the applicability of CTA, CTM, CBO, and RFC in predicting fault-prone modules. The authors take 2.0 version of Eclipse. Performance of Univariate Binary Regression is evaluated using Odds ratio. The results deduce the acceptable predictive performance of coupling metrics. The next experiment is performed using collinearity analysis, wherein the results remain the same.

Yuming and Hareton [35] aim to assess the impact of CK metrics in identifying the fault-prone modules in KC1 dataset. The study is carried out in on ordinal variables, i.e, the severity of faults. Four statistical and tree-based algorithms are employed wherein Precision, correctness, and completeness are used as a performance indicator. Results in all the cases are quite consistent, i.e, the coupling metrics are very productive in predicting the severity level of any software module.

Aggarwal *et al.* [34] perform an experiment on 12 different software system developed by student. The CK metrics suite and Briands metrics suite are used as independent variables, whereas binary fault information is used as a dependent variable, which is collected from the testing team. The results of Bivariate logistic regression is evaluated on the scale of Sensitivity and Specificity. The results infer the significant predictive ability of CBO, RFC, while metrics in the Briands' metric suite could not show significant results.

Olague *et al.* [57] empirically evaluate three object-oriented metric suites (CK, MOOD, and QMOOD) in predicting faults on six Rhino versions. Using bivariate correlation between metrics and defects they conclude RFC as strongly correlated with software defects, while CBO has minor to moderate correlation. Next, by using logistic regression analysis, RFC is found significant in all six versions of Rhino, whereas CBO is found significant in five versions of Rhino.

Pai and Bechta [56] investigate how Bayesian methods can be used for assessing software fault content (number of faults) and the presence of faults. They use KC1 dataset to analyze the correlation between CK metric suite with faults. They conclude that CBO, RFC, and SLOC are very significant for assessing both fault content(number of faults) and the presence of faults.

Goel and Singh [33] assess the impact of three coupling metrics, CBO, Fan-in, and RFC on KC1. The dataset comprises the information of 145. Univariate logistic regression is used to figure out the relationship between selected coupling metrics and a dependent binary fault variable. The result is in the favor of CBO and RFC in predicting the fault-prone classes in KC1 dataset.

Jie *et al.* [55] assess the usefulness of CBO and RFC on NASA's KC1 dataset and conclude the effectiveness of both metrics using Correlation and Regression analyses. However, their third experiment using Neuro-fuzzy approach results in the effectiveness of both metrics.

Shitnawi and Li [32] aim to evaluate the productiveness of CK coupling metrics along with CTA, and CTM. For the experimentation, three versions of Eclipse are utilized. Fault label is taken as a dependent variable, which is collected Bugzilla fault repository. In the first experiment, Univariate Binary Logistic Regression is employed in predicting faulty modules. While in the second experiment wherein Univariate Multinomial Regression is used in predicting faults' level of severity. Which is computed by number of faults reported in each module. In both of the experiments, the acceptable level of predictive performance of included metrics is reported.

Zimmermann and Nagappan [54] assess the dependency factor in predicting *fp* binaries in Windows Server 2003. The dependency factor includes call dependencies, data dependencies, and dependencies specific to Windows. Binary refers to Portable executables, COMs, or DLLs. Call dependency includes import calls, export calls. The dataset comprises 2252 binaries. A dependency graph is generated using MaX and defect data is collected using the post-release defect achieve maintained by Microsoft. Prediction is done for classification and ranking (number of defects). They evaluated CCM, Nagappan's CyclicClassCoupling, Fan-in, and Fan-out along with some non-coupling metrics.

Aggarwal *et al.* [23] disclose the association of coupling metrics with that of fault detection. The study is in fact the replication of another study performed by Birnad *et al.* [38, 39]. 10 coupling metrics are using as an independent variable while binary fault information is used as the dependent variable. Initially, every independent variable is assessed individually using univariate logistic regression. Later the significant metrics are employed together using multivariate logistic regression. The experiment is carried out on 12 different software systems developed in Java by the students. In the first experiment metrics which are related to coupling show an acceptable level of impact on the presence of faults. However, the metrics that are related to export coupling alone cannot produce a satisfactory results. In the second experiment, it is found that when coupling metrics are used in combination of size metrics, the predictive performance improves.

Kpodjedo *et al.* [52] investigate the fault predictive ability of CK metric suite and their proposed ECGM metrics. In addition to ECGM, the most accurate model is the one built on CBO and RFC inclusively.

English *et al.* [53] evaluate the usefulness of CK metric suite using Bugzilla reports and CVS commits of two software products Eclipse JDT and Mozilla. The authors use univariate and multivariate logistic regression to assess the impact of individual metrics and LOC with software faults. They report high correctness values of RFC and CBO. Next, in linear regression modeling, RFC and CBO are found reasonable predictors of software faults. Finally, they give a verdict that LOC along with CBO and RFC are the best predictors of *fp* classes.

Selvarani *et al.* [31] exclusively assess the productivity of RFC in predicting software faults. Multifunctional estimation model is used to map their relationship on five software products developed in Java. The results show that there exists an influence of 3% with the value of 24, which becomes 24% with the value of 93. Moreover, there is a clear indicator of occurrence of fault when the value of RFC is above 100.

Jureczko and Spinellis [50] develop a regression model for predicting faults using CK metric suite and LOC. They use five proprietary and eleven open-source projects. In the process of eliminating the least correlated metrics, they drop RFC, while keeping CBO.

Malhotra *et al.* [30] aim to evaluate CK metrics suite in predicting fault classes in KC1 dataset. This time severity of fault is used as a target label. Initially, correlation is used to figure out the association between metrics. In the second phase, RFC and CBO are tested for their fault predictive ability using a support vector machine (SVM) wherein radial basis is used as a kernel function. The results of Sensitivity, Specificity, Precision, Completeness, and AuC advocate the viability of both coupling metrics (CBO and RFC) in the identification of severe fault carrier classes.

Shatnawi [51] investigate the acceptable risk level using CK metric suite. Two versions of Eclipse 2.0 and 2.1 are taken as a case study. Modeling is done through univariate logistic regression. CBO and RFC are found significant predictors of faults at the 95% confidence level.

Elish *et al.* [48] evaluated three OO metric suite that comprises six coupling metrics, i.e., Ca, Ce, I, CF, RFC, and CBO. Data is collected from three versions of Eclipse. The authors evaluate individual metrics using Spearman correlation coefficients and report the significance of Ca, Ce, CBO, and RFC, while insignificance of CF and I.

Johari and Kaur [29] carried out an experiment on JHotDraw 7.5.1, which is an open-source software system developed in Java. CK metrics suite is used as an independent variable, which is computed using ckjm parser. The fault label is determined by reviewing the description of revisions. A total of 613 instances are collected. The empirical evaluation is conducted in three different dimensions. In the first experiment, linear correlation is assessed between bug count and the included independent variables, wherein a strongly positive correlation is observed with WMC and RFC. Moreover, an average level of correlation is observed between LCOM and CBO. In another empirical test, predictive ability of RFC, and CBO independently with fault count using Univariate linear regression model. Value of R^2 infers the positive performance of both of the coupling metrics, i.e., CBO and RFC.

Rathore and Gupta [49] evaluate 19 class level metrics (including coupling metrics) on five publicly available project datasets. The authors first evaluate each metric independently using univariate logistic regression. Next, the correlation between metrics is computed, where strongly correlated metrics are dropped and the remaining subset of metrics are evaluated using multiple releases of the same software. In their first experiment, they conclude that CBO, RFC, import, and export coupling metrics are significantly correlated with software fault in four datasets.

He *et al.* [47] aim to build simplified metric set for SFP. They take 34 releases of 10 open-source projects from PROMISE repository. Model building is done using J48, LR, NB, DT, SVM, and BN. Independent variables are CBO, RFC, Ca, Ce, and CBM and the dependent variable is Binary. They first select TOPK metrics for their experiment, wherein CBO, RFC, and Ce are selected.

Kumari and Rajnish [46] propose a class level complexity metric (CLCM). Their objective is to evaluate the performance difference of CLCM and some other coupling metrics CBO, RFC, MPC, LMC, Fan-out, and EXT. Dataset is collected from three versions of Eclipse 2.0, 2.1, and 3.0 and the experiment is performed on each version independently. Two types of labels are used binary and Severity level (Minimum, Low, Medium, and High). For both types of dependent variables, Spearman correlation coefficients and univariate logistic regression are used to investigate the impact of a metric on SFP. The results of this experiment show the strong correlation of coupling metrics with faults, and classification accuracy for all coupling metrics lies between 0.70 to 0.75. More specifically EXT, MPC, and RFC have the strongest impact on pre-release faults.

Anwer *et al.* [28] evaluate the predictive performance of three coupling metrics, Ca, Ce and CBO. The prediction model is built using multivariate linear regression. They use seven different size open-source projects which are developed in Java. These datasets are Jedit, Camel, Arc, Ant, Velocity, Xerces, and Poi. They conclude that Ce has a better correlation with faults than Ca and CBO. They report that CBO and Ce are strongly correlated in all the seven selected projects.

Kumair *et al.* [209] consider CBO, RFC, Ce, Ca, CBM, WMC, DIT, NOC, LCOM, NPM, LOC, LCOM3, DAM, MOA, MFA, IC, CAM, AMC, Max-CC, Avg-CC as independent variable and presence and absence of fault a dependent variable. They take 31 projects from Promise repository, which are developed Java. Upon applying the multiple statistical tests (i.e.Chi-squared test, Gain ratio feature evaluation, OneR, Feature evaluation, Univariate Logistic regression, Principle component analysis) they reported the outperformance of OO metrics in SFP.

Bosco *et al.* [210] introduced Clang/LLVM-based static analysis tool for identifying the software quality threshold for software product metrics. Initially, the authors employ the compiler infrastructure and introduce a Clang AST tool to gather Abstract Syntex Tree(AST)-based metrics. Later, the statistical analysis has been conducted to identify the reference thresholds of 22 code metrics including Fan-in and Fan-out. Wherein, acceptable threshold of Fan-in is less than 8. However, the underline identified thresholds are sensitive to probability distribution and thus could change by varying the distribution. Moreover, the inclusion of more libraries could alter the presented findings. Finally, the thresholds are built using the identified code smells. Therefore the threshold may not perform well in the unseen data.

Deepak *et al.* [211] aim to classify faults and to explore the usability of Factor Analysis with Regression. The constructed models used to estimate the proneness of faults surpasses Multivariate linear regression. CK+OO metrics are utilized for the experimentation and performance has been evaluated using R Square and Adjusted R Square and AEEEM datasets have been taken as a case study. The results support the inclusion of CK and OO metrics to be utilized for SFP.

Malhotra *et al.* [212] provide an effective defect prediction framework for imbalanced data by employing cost-sensitive classifiers and stable performance measures like GMean, Balance, and AUC. The authors utilized three Apache projects and four decision tree-based classifiers, i.e, J48, AdaboostM1, Bagging, and RSS. The Independent variables of the study are CBO, RFC, Ce, Ca, MFA, CBM along with

14 object-oriented metrics. Friedman and Wilcoxon signed-tank test supports the effective performance of included metrics.

Navneet *et al.* [213] employs CBO, RFC, along with other metrics of CK metrics suite to identify the dichotomy that can effectively distinguish the *fp* and *nfp* classes. The authors analyzed numerous threshold-based techniques to determine the most effective one. The results reported the significantly effective performance of concordance probability and maximum sum of sensitivity and specificity. Moreover, it was concluded that WMC, CBO, RFC, LOC, and NPM were statistically significant predictors and can be used for the threshold identification process.

Navneet *et al.* [214] performed a study wherein WMC, CBO, RFC, Ce, and LoC utilized over commonly used metrics for SFP problem. The authors proposed fuzzy cognitive map to eliminate data dependency for software fault prediction. The proposed cognitive map was learned from experts without the need or dependency on a prior project dataset wherein the technique provides more plausible and accurate decisions in predicting the faulty modules.

Sushant *et al.* [215] conducted 880 experiments to analyze the variation in the performance of 10 SFP models by concerning the class imbalance problem. The authors used 22 public datasets consisting of 41 software metrics comprising numerous coupling metrics. Moreover, 10 baseline SFP methods, and 4 sampling techniques are also examined. Performance has been evaluated using F-measure. The results conclude that Random Sampling techniques give the best result with most of the ML algorithms.
Table 3.1 and 3.2 summarize the studies that explicitly address the coupling metrics. The first column of Table 3.1 show the reference of study, 2^{nd} and 3^{rd} columns show the coupling and non-coupling metrics used in the study. The 4^{th} column shows the type of dependent variables used. The dependent variable has three possible values;

1. **Binary** shows *fp* and *nfp* labels

2. **Ordinal** shows fault severity

3. **Numerical** shows number of faults

Table 3.1: Summary of the independent and dependent variables used in the studies

Study	**Coupling metrics**	**Non-coupling metrics**	**Type of Dependent variable**
[19]	X[1-7, 19-21]	X [8 - 18]	Numerical
[206]	Fan-in, Fan-out	LoC, CC	Numerical
[45]	CBO, RFC, FunctCall	Intercept, MaxStatNest, FunctDef, DIT, NOC, LCOM, WMC	Binary
[44]	CBO, NSSR, NCC, CDM, Fan-in, Fan-out, RFC	LoC, WMC, DIT, CHNL, NOC, NOD, NCIM, WIH, HIH	Numerical
[43]	Fan-in, Fan-out, CDM, OSC	CC, LoC	Numerical
[117]	Fan-in, Fan-out, CDM, OSC	CC, LoC	Numerical
[42]	Fan-in, Fan-out, CBO, NCC, NSSR, CDM, RFC	WMC, DIT, CHNL, NOC, NCIM, WIH, HIH, CC, LOC, NOD, No. of global variables, No. of clients	Numerical
[41]	CBO, RFC, MPC, ICP, NIHICP, DAC, OCAEC, OCMEC, OMMEC, OMMIC, OCAIC, OCMIC, IFCAIC, IFCMIC, IFMMIC, FCAEC, FCMEC, FMMEC,IFMMIC, FCAEC	NMO, SIX, NMA, LOC, WMC, DIT, AID, NOA, NOP, NMI, NOC, NOD, CLD, ACAIC, DCAEC, ACMIC, DCMEC, AMMIC, DMMEC	Binary

[192]	CBO, NAS	None	Numerical
[40]	CBO, RFC, OCAEC, OCMEC, OMMEC, OMMIC, OCAIC, OCMIC, IFCAIC, IFCMIC, IFMMIC, FCAEC, FCMEC, FMMEC	LCOM, SLOC, WMC, DIT, ACAIC, DCAEC, ACMIC, DCMEC, AMMIC, DMMEC	Binary
[59]	CBO, RFC	DIT, NOC, WMC, IC, CBM, NOMA, AMC, IC, CBM, NOMA, AMC	Numerical
[39]	CBO, RFC, MPC, ICP, NIHICP, DAC, OCAEC, OCMEC, OMMEC, OMMIC, OCAIC, OCMIC	NMO, SIX, NMA, LOC, WMC, DIT, AID, NOA, NOP, NMI, NOCAMMIC, DMMEC	Binary
[38]	CBO, RFC, MPC, ICP, NIHICP, DAC, OCAEC, OCMEC, OMMEC, OMMIC, OCAIC, OCMIC, IFCAIC, IFCMIC, IFMMIC, FCAEC, FCMEC, FMMEC	NMO, SIX, NMA, LOC, WMC, DIT, AID, NOA, NOP, NMI, NOC, NOD, CLD, ACAIC, DCAEC, ACMIC, DCMEC, AMMIC, DMMEC,	Binary
[58]	CBO, RFC, OCAEC, OCMEC, OMMEC, OMMIC, OCAIC, OCMIC, IFCAIC, IFCMIC, IFMMIC, FCAEC, FCMEC, FMMEC, NPAVG,OCMEC	SIX, LCOM, SLOC, WMC, DIT, ACAIC, DCAEC, ACMIC, DCMEC, AMMIC, DMMEC, NMA, NMO	Binary

[21]	OCAIC, OCAEC, OCMIC, OCMEC,	ACAIC, ACMIC, DCAEC, DCMEC, DIT, NOC	Binary
[37]	CBO, RFC, Fan-in	NMC, LOC, LCOM, DIT, NOC	Binary
[207]	CBO, RFC	DIT, LCOM, NOC, NOM, SLOC	Numerical
[6]	RFC, CBO,	WMC, DIT, LOC, LCOM, NOC, LCOMN	Binary and Numerical
[147]	CBO, RFC	DIT, LCOM, NOC, NOM, SLOC	Numerical
[36]	CTA, CTM, CBO, RFC	WMC, DIT, NOC, NOAM, NOOM, NOA, NOO	Binary
[35]	CBO, RFC	WMC, DIT, NOC, LCOM, SLOC	Binary
[208]	CBO, RFC, Fan-in	PPD, ATPD, CBO, DIT, LCOM, NOC, RFC, WMPC, and DOC	Numerical
[34]	CBO, RFC, FCAEC, FCMEC, FMMEC, IFCAIC, IFCMIC, IFMMIC, OCAEC	LCOM1, LCOM2, NOC, DIT, WMC, ACAIC, DCAEC, ACMIC, DCMIC, DCMEC, AMMIC	Binary
[57]	CBO, RFC	DIT, LCOM, NOC, WMC, MC, AHF, AIF, MHF, MIF, CIS, DAM, DCC, MFA, NOM	Numerical
[56]	CBO, RFC	WMC, DIT, NOC, SLOC, LCOM	Binary and Numerical
[33]	CBO, Fan-in, RFC	DIT, LCOM, WMC, NOC, CC, SLOC, NMC	Binary and Numerical
[55]	CBO, RFC	WMC, DIT, NOC, SLOC, LCOM	Numerical

[32]	CTA, CTM, CBO, RFC	WMC, DIT, NOC, NOAM, NOOM, NOA, NOO	Ordinal (Severity)
[54]	Fan-in, Fan-out	LOC, No. of parameters, CC, NOM, SubClasses DIT, ClassCoupling, CCC	Binary and Numerical
[23]	CBO, RFC, DAC, MPC, ICP, NIHICP, FCAEC, FCMEC, FMMEC, IFCAIC, IFCMIC, IFMMIC, OCAEC, OCAIC, OCMIC, OCMEC, OMMEC, OMMIC	IHICP, ACAIC, DCAEC, ACMIC, DCMEC, AMMIC, DMMEC, LCOM1, LCOM2, LCOM3, TCC, LCC, ICH, NOC, DIT, CLD, NOP, NOD, NOA, NMO, NMI, NMA, SIX, AID, NA, NM, WMC, PM, NPM, NPARA, LOC	Binary
[52]	CBO, RFC	WMC, DIT, NOC, LCOM, EC, CR, LOC	Binary and Numerical
[53]	CBO, RFC	WMC, DIT, NOC, LOC	Binary
[31]	CBO, RFC	WMC	Binary and Numerical
[50]	CBO, RFC, Ca, Ce	CBM, WMC, DIT, NOC, LCOM, LCOM3, NPM, DAM, MOA, MFA, CAM, IC, AMC, CC, LOC	Binary
[30]	CBO, RFC	WMC, DIT, NOC, LCOM, SLOC	Binary
[51]	CBO, RFC	WMC, DIT, NOC	Binary
[48]	Ca, Ce, CBO, RFC	NC, I, D, AHF, MHF, AIF, MIF, CF, PF, WMC, LCOM, DIT, NOC	Numerical
[29]	CBO, RFC	WMC, DIT, NOC, LCOM, Token count, WMC(CC)	Numerical

[49]	CBO, RFC, CA, CE,	WMC, DIT, NOC, IC, CBM, MFA, LCOM, LCOM3, CAM, MOA, NPM, DAM, AMC, LOC, CC	Binary
[47]	RFC, CBO, CA, CE	CBM, WMC, DIT, LCOM, NOC, DAM, NPM, MFA, CAM, MOA, IC, AMC, LCOM3, MAX CC, AVG CC, LOC	Binary
[46]	RFC, MPC, CBO	NOS, UWCS, CC, NLOC, EXT, LMC, TCC, PACK, NOM, LOM2, INST, MAXCC, FOUT, AVCC, CLCM	Binary, Multinomial
[28]	Ca, Ce, CBO	None	Numerical
[209]	CBO, RFC, Ce, Ca	CBM, WMC, DIT, NOC, LCOM, NPM, LOC, LCOM3, DAM, MOA, MFA, IC, CAM, AMC, Max-CC, Avg-CC	Nominal

Table 3.2: Summary of the methodologies and datasets used in the studies

Study	**Dataset quantity/-name**	**Prog. lng.**	**Availability**	**Analysis methodol-ogy**	**Performance Measure**
[19]	73/Industrial systems	-	Private	MLR	Regression coefficients
[206]	1/Com. system	-	Private	Scatter plot	Visual analysis

[45]	8/Information management systems	C++	Private	Univariate and multivariate logistic regression	Regression coefficients, p-value, Accuracy
[44]	8/patient collaborative care system	C++	Private	Spearman correlation	Correlation coefficients
[43]	1/OASIS	COBOL	Private	Spearman correlation	Correlation coefficients
[117]	1/OASIS	Java	Public	Spearman correlation	Regression coefficients
[42]	4/OASIS, Ffortid, Collaborative Care System, Electronic File Transfer Facility	COBOL, C++, C, Java	Private	Pearson correlation coefficients	Correlation coefficients
[41]	8/Information system	C++	Private	Univariate logistic regression	Regression coefficients
[192]	5/GNU, LEDA, SEG1, SEG2, EFOOP2	C++	Private	Spearman correlation	Correlation coefficients
[40]	1/telecom system	C++	Private	Logistic Regression	regression coefficients andR2
[39]	8/Students' projects	C++	Private	Logistic regression	Completeness, Correctness, Kappa, R2

[59]	3/Three subsystems of Human Machine Interface,	C++	Private	Univariate logistic regression	Coefficients, The statistical significance(p-value)
[38]	1/LALO 1.3	C++	Private	Univariate Logistic regression	regression coefficients and standard error
[58]	1/Telecom. framework	C++	Private	Pearson correlation	Regression coefficients
[21]	2/Commercial Application 2.5 and 2.6	Java	Private	Univariate logistic regression	Regression coefficients, R2 coefficients
[37]	1/Managing system	Java	Private	Univariate linear regression	multiple coefficients of determination, R2 coefficients
[207]	1/E-commerce application	C++ and Java	Private	Correlation between metrics, Cook's distance	Coefficient
[6]	7/Mozilla's Versions	C++	Partial	Univariate logistic regression, C4.5 , Neural Network	R2 coefficients, Precision, Correctness, Completeness

[147]	5/Telecom. Apps	C++	Private	Correlation coefficient, Poisson regression, Negative binomial regression	Correlations, dispersion coefficients, and Alberg diagrams.
[36]	1/Eclipse 2.0	Java	Partial	Univariate Binary Analysis	Odds ratio
[35]	1/KC1	C++	Public	Univariate Logistic regression, Bayes network, RF, Nearest neighbor	Regression coefficients, Precision, Correctness, Completeness
[208]	1/KC1	C++	Public	Stepwise linear regression	Standard Error, Sum of square error
[34]	12/Students' Projects	Java	Private	Univariate logistic regression	Sensitivity, Specificity
[57]	6/Rhino	Java	Private	Spearman correlation between variables, Spearman correlation, ULR, MLR	Coefficient

[56]	1/KC1	C++	Public	Spearman correlation analysis, ordinary least-squares (OLS, BLR, and Bayesian Poisson regression (BPR)	Adjusted coefficient of determination, Deviance information criterion,
[33]	1/KC1	C++	Public	Univariate logistic regression, Multivariate Logistic regression, ULR, MLR	R^2 coefficients
[55]	1/KC1	C++	Public	Correlation, ULR, MLR, and Neuro-fuzzy	Coefficient, Constant, R^2, P-value
[32]	3/Eclipse 2.0, 2.1, and 3.0	Java	Partial	Univariate Binary logistic Regression, ULR	Regression coefficients
[54]	1/Windows Server 2003	C	Private	ESCROW analysis, Spearman correlation, PCA, LR	R^2, Adjusted R^2, Pearson correlation, Precesion, Recall

[23]	12/Students' Projects	Java	Private	Univariate logistic regression	coefficients, standard error
[52]	1/Rhino 1.6R5	Java	Partial	Linear Regression, logistic regression, Regression tree	—
[53]	2/Eclipse JDT and Mozilla	Java, C++	Partial	Univariate and Multivariate logistic regression	Coefficient, Constant, p-value, R^2
[31]	5/-	Java	Private	Multifunctional estimation	Defect proneness index
[50]	16/versions of 11 software products	Java	Public	T-test statistic	coefficient
[30]	1/KC1	C++	Public	SVM	Sensitivity, Specificity, Precision, Completeness, AuC
[51]	2/Eclipse 2.0 and 2.1	Java	Partial	Univariate logistic regression	p-value, VARL values,
[48]	3/Eclipse 2.0, 2.1, and 3.0	Java	Partial	Spearman correlation coefficients	Coefficient

[29]	1/JHotDraw 7.5.1	Java	Partial	Correlation coefficients and ULR	Correlation coefficients and Squared error
[49]	16 releases of 5 software products	Java	Public	ULR, correlation coefficient, and MLR	Coefficient, p-value, and Odd ratio
[47]	34 releases of 10 softwares product-s/Promise repository	Java	Public	J48, Logistic Regression, Naive Bayes, Decision Table, Support Vector Machine, Bayesian Network, Analysis of variance	Precision, Recall, F Measure
[46]	3/Eclipse 2.0, 2.1, and 3.0	Java	Partial	Spearman correlation coefficients and Binary/MLR	R^2 coefficient, AuC
[28]	7/Jedit, Camel, Ant, Arc, Velocity, Xerces, Poi	Java	Public	Spearman correlation	Regression coefficients
[209]	31/Promise repository	Java	Public	Chi-squared, Gain ratio, ULR, PCA	F-Measures, Accuracy

3.2.4 Analysis of Coupling Metrics w.r.t Their Usage

The coupling metrics discussed in Section 3.1 are searched for their exclusive usage and usefulness in SFP studies. Table 3.3 shows the mapping of coupling metrics against the corresponding studies. "Not used exclusively" shows that the respective metric has not been evaluated exclusively in SFP.

TABLE 3.3: Coupling metrics w.r.t. evaluating studies

Metric	SFP Studies on respective metric
Fan-in	[17, 18, 33, 37, 42–44, 54, 117, 206, 208]
Fan-out	[17, 18, 42–44, 46, 54, 117, 206]
GIF	[17, 18]
LIF	[17, 18]
C∼	[188]
DCBO	Not used exclusively
DC	Not used exclusively
FM	Not used exclusively
CBO	[6, 23, 28–42, 44–53, 55–59, 147, 192, 207–209]
RFC	[6, 23, 29–42, 45–53, 55–59, 147, 208, 209]
CtrlC	Not used exclusively
DataC	Not used exclusively
E	Not used exclusively
MR	Not used exclusively
SC	Not used exclusively
OCM	Not used exclusively
CCM	[54]
OM	Not used exclusively
DAC	[23, 32, 36, 38, 39, 41]
MPC	[23, 32, 36, 38, 39, 41, 46]
COF	[48]
NIHICP	[23, 41]
ICP	[23, 38, 39, 41]

DM	Not used exclusively
Ca	[28, 47–50, 209]
Ce	[28, 47–50, 209]
I	[48, 54, 58, 117]
CDM	[42–44, 117]
OSMC	[42, 117]
Briand suite	[21, 23, 34, 38–43, 58]
NAS	[192]
ABC	Not used exclusively
DCC	[57]
CFS	Not used exclusively
Nagappan suite	[54, 57]
WTCoup	Not used exclusively
AM	Not used exclusively

The included studies depict that,

1. Coupling metrics in general, and CBO, RFC, Fan-in, and Fan-out are specifically found useful in predicting software faults, irrespective of the dataset size, type of dependent variable.
2. Fifteen coupling metrics discussed in Section 3.1.3, have not been addressed by the SFP community.
3. Every time, coupling metrics are evaluated individually. Therefore, the combined effect of multiple coupling metrics is hidden.
4. Coupling metrics are evaluated using products developed in C, C++, Java, or COBOL.
5. Mostly, statistical techniques (i.e., Linear and Logistic regression, etc) are used for modeling, except for few studies [6, 30] wherein Decision tree, Neural network, SVM, Bayesian Learners are used.

6. A wide variety of datasets are used by the research community which includes Public, Partial, and Private. Likewise, all type of labels is used including Binary, Numerical and ordinal (severity and range of faults).

3.3 Studies on the Coupling Levels/Principles

Software modules' coupling is indispensable. Absolute exclusion of coupling implies the existence of only one module. Thus it would be difficult to understand, develop, test, and deploy. Comprehensible level of coupling is always desirable, however, when there exists complexity in coupling in terms of levels and volume of data flow there comes incomprehensibility. Consequently, the programmer fail to take into account all necessary associated flows. This lack of comprehension cause him to commit/induce faults in a program module. Hence it may be used with great care. This spurs to discover/disclose maximum possible levels associated to coupling that segregate desirable coupling from that of undesirable coupling.

3.3.1 Inclusion Criteria

The research literature is quite rich on the discussion of coupling levels and principles, however we include studies that having following properties.

1. Software coupling is presented as a framework
2. Different principles/levels of coupling are presented

3.3.2 Exclusion Criteria

The studies that discuss coupling without having any concern about desirable and undesirable aspects of coupling are not considered.
Example of such study is [216], wherein three principles are stated; Stable Dependencies Principle, Acyclic Dependencies Principles, and Stable Abstraction

Principle. Likewise, in another article by Booch *et al.* [217], wherein different types of relatedness between classes of software system are discussed.

3.3.3 Brief Description of the Studies Included

Yourdon and Constantine [118] are the first authors (to the best of our knowledge), who enumerate five basic rules of coupling. According to them, few couplings are desirable, i.e, narrow, direct, local, obvious, and flexible. In contrast to that few couplings are undesirable, i.e, broad, indirect, remote, hidden, and rigid.

Myers [24] defines six levels of coupling and scales them from lowest level of coupling (best) to highest coupling (worst). These levels are Data, Stamp, Control, External, Common, and Content. Moreover, the authors reported that Data coupling is low susceptible to errors, while stamp and control are medium susceptible to errors whereas the last three levels are highly susceptible to errors. The levels are also mapped against usability and extensibility.

P. Jones [25] presents three broad categories of coupling; Normal coupling, global coupling, and content coupling. Normal coupling includes Data, Tramp, Stamp, Bundling, Control, and Hybrid coupling. The authors later map these levels to the degree of susceptibility of ripple effects, modifiability, understandability, and usability. Global and content coupling are declared as the most susceptible to ripple effects, carry the least understandability, and have bad usability.

Offutt *et al.* [26] are more focused on extending the coupling levels by further granulating the coupling levels proposed by Myers [24]. They argued that previously, coupling levels have been defined subjectively and are not quantified. They develop a general software metric system of coupling. They precisely defined the levels of coupling, incorporated the notion of direction into the coupling levels. Later, they implemented their metric system so that it measures the coupling between pairs of procedures written in C programs.

Dhama [197] specify, implement, and verify quantitative model for software coupling. The author divides coupling into four categories (data, control, global, and environment coupling) followed by the quantification of each category in his proposed metric. He further uses $\alpha, \beta, and \gamma$ as proportionality constants, associated with control factors. These factors are given a value of 2 as a heuristic estimate.

Timothy and Laganiere [27] are more inclined towards coupling levels in the OO paradigm. They elaborate the coupling levels using short examples from Java language and introduce two new coupling levels, Inclusion/Import, and Type use coupling.
Table 3.4 shows the list of coupling levels and principles proposed in the included studies. The levels are sorted by increasing order of their desirability assigned by the corresponding authors.

3.3.4 Shortlisted Coupling Levels

The accumulated list of coupling levels by all the authors discussed earlier cannot be taken as a whole. The reason is that few coupling levels are associated with few avoidable properties. These are briefly discussed below;

1. Few coupling levels cannot be counted as coupling levels, like Inclusion/Exclusion coupling and Independent coupling

2. Few coupling levels do not fall under the scope defined for this study, like, External coupling, hybrid coupling, and Type use coupling. The reason is that these coupling levels do not take into account the coupling between two *software* modules.

3. Few coupling levels are derived from some other base coupling levels, like Stamp data/control coupling, has been derived from Stamp coupling and Control coupling.

4. Few coupling levels are common in name and definition. Like Content coupling defined by P. Jone's is same in definition as that of Content coupling

Table 3.4: Coupling levels proposed in different studies

Levels					**Principles**				
Myers	**P. Jones**	**Offutt *et al.***	**Dhama**	**Timothy**	**Yourdon and Constantine**				
Content	Content	Tramp	Environment	Content	Volume	Indirect	Remote	Hidden	Rigid
Common	Common	Global	Control	Common	Narrow	Direct	Local	Obvious	Flexible
External	Database	Non-local	Global	Control					
Control	Hybrid	External	Data	Stamp					
Stamp	Control flag	Stamp data/control		Data					
Data	Descriptive flag	Scalar data/control		Routine call					
	Stamp	Stamp control		Type use					
	Bundling	Scalar control		Inclusion/import					
	Data	Stamp data		External					
	Tramp	Scalar data							
	Zero scale	Call							
		Independent							

proposed by Myers.

By considering a few or all of the above reasons, we do not take into account a few coupling levels. The following subsection briefly describes the coupling levels included in our study by the decreasing order of their complexity reported by earlier studies.

1. *Content coupling* shows the access of components of one module to the components in another module. In the programming languages, *goto* statement is used for this purpose. Although modern programming languages prohibit the use of content coupling. However, there are few hacks for employing it[27].

2. *Common coupling* establishes between modules when both of the modules assess(read, write) common global variables. Common coupling has another name *global coupling*. Moreover, there is another coupling level proposed in [118], i.e., remote coupling is also based upon common coupling. Liguo *et al.* conduct detailed research on common coupling and its various dimensions [218].

3. *Control coupling* or *Control flag coupling* is established between two software modules when a module directs the control of another module by passing parameters. Scalar control coupling is also a type of control coupling, wherein passed parameter is a primitive data type. Moreover, stamp control coupling is also a type of control coupling, wherein passed parameter is a data structure. Besides these two types, there exists another type name as *Hybrid coupling*. which is also a part of control coupling.

4. *Descriptive flag coupling* is similar to that of *control coupling*, discussed earlier. The only difference is that in the control flag coupling the return value is of Bool type, whereas in descriptive flag coupling the return type can be of any type other than Bool.

5. *Stamp coupling* establishes when one module sends/receives complete data structure (object, interface, array, etc) to/from another module. According

to this definition, both Stamp coupling and stamp data coupling are the same.

6. *Data coupling* establishes between two modules when one module sends/receives primitive data to/from another module. According to this definition both *Data coupling* and *scalar data coupling* are the same. There are few other types also, which are used in this context, i.e, *Bundling*, *Tramp coupling*, etc.

7. *Call coupling* implies no coupling between software modules. In this type of coupling neither any parameter is passed nor any data is returned. It is also named as zero scale coupling and routine call coupling[25].

Out of the five properties of worst coupling defined by Yourdon and Constantine [118], two properties, i.e, remote and indirect coupling are not considered in this study. The reason is that remote coupling is similar to global coupling while indirect coupling is in fact two levels of coupling, which does not fall under the scope of our study. Keeping in view this, we consider only three principles of coupling in our comparison and evaluation. These principles are broad, hidden, and rigid coupling.

3.3.5 Analysis of Coupling Metrics w.r.t. Coupling Levels

In this section, we aim to assess the coupling metrics for the presence and discrimination of coupling levels (discussed in the last section). Keeping in view the important and disjoint coupling levels, there are a total of nine coupling levels. In addition to that, we aim to assess the coupling metrics if they take into account the coupling principles or not. Table 3.5 shows the coverage of the coupling metrics to the coupling levels shortlisted earlier. We classified such coverage into three types, i.e., None, Partial, and Full shown by ✗, ◐, and ✓ respectively see Table 3.5).

1. If the value of metric remains the same by the presence and absence of coupling level, then it is shown by ✗.

2. If the value of metric remains the same by the presence and absence of coupling level but does not segregate between different levels, then it is shown by ◐. The example of such mapping is observed in Fan-in metric. The reason is that the value of Fan-in value reflects the presence of data coupling, however, it does not change by changing the coupling level from data coupling to control coupling. Such segregation is very vital. The reason is that complexity associated with each coupling level significantly varies, this is expected to be addressed by the coupling metrics also.

3. If the value of coupling metric changes by the presence/absence of coupling level and along with the representation of varying coupling level, then it is shown by ✓. This state is the best of all the three states, just discussed.

Principles' coverage includes three principles of Yourdon and Constantine [118]; volume, hiddenness, and rigidness. ✓and ✗show the presence and absence of the corresponding factor in the respective metric.
Shaded rows show the metrics which are not yet used in SFP, exclusively. Table 3.5 depicts that,

1. All the coupling metrics fail to incorporate three important aspects of coupling, i.e., Content coupling, hiddenness, and rigidness.

2. There are a total of 27 coupling metrics that partially incorporate data-related coupling like (Control, Stamp, Data, and Stamp).

3. Very few metrics successfully incorporate Control, Descriptive, Stamp, and Data couplings.

4. None of the coupling metrics is able to incorporate all the four levels of coupling.

5. Data flow volume is an important coupling parameter that is not addressed by about thirty coupling metrics.

TABLE 3.5: Coverage of coupling levels and principles by Coupling metrics

Metric	**Levels**							**Principles**		
	Content	**Common**	**Control**	**Descriptive**	**Stamp**	**Data**	**Call**	**Volume**	**Hidden**	**Rigid**
Fan-in	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
Fan-out	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
GIF	✗	◐	✗	✗	✗	✗	✗	✓	✗	✗
LIF	✗	◐	✗	✗	✗	✗	✗	✓	✗	✗
C∼	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
DCBO	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
DC	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
FM	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
CBO	✗	✗	✗	◐	◐	◐	◐	✗	✗	✗
RFC	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
CtrlC	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
DataC	✗	✓	◐	◐	◐	◐	◐	✓	✗	✗
E	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
MR	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
SC	✗	◐	◐	◐	◐	◐	◐	✓	✗	✗
OCM	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
CCM	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
OM	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
DAC	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
MPC	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
COF	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
NIHICP	✗	✗	◐	◐	◐	◐	◐	✓	✗	✗
ICP	✗	✗	◐	◐	◐	◐	◐	✓	✗	✗
DM	✗	✓	✓	✗	✓	✓	✗	✗	✗	✗
Ca	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
Ce	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
I	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
CDM	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
OSMC	✗	✓	✓	✓	✓	✓	✗	✗	✗	✗
Briand suite	✗	✗	◐	◐	◐	◐	✗	✗	✗	✗
NAS	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
ABC	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
DCC	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
CFS	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
Nagappan suite	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗
WTCoup	✗	✗	◐	◐	◐	◐	◐	✗	✗	✗
AM	✗	◐	◐	◐	◐	◐	◐	✗	✗	✗

6. The simplest of all the coupling is call coupling, which is not considered by nine of the coupling metrics.

7. The worst of all the coupling metrics in maximum ignoring the important aspects of coupling are CDM and DAC.

8. The best of all the coupling metrics in maximum incorporating the important aspects of coupling are FM, OM, and OSMC.

It can safely be stated that the complexity associated with the coupling cannot be enveloped by any of the coupling metrics as a whole. Therefore, there is a need to propose either a new measure of coupling or augment any of the existing measures of coupling that incorporates the yet missing aspects of coupling. We believe that if we are successfully able to incorporate these aspects in coupling metrics and used in SFP, this would potentially assist the software testing community.

3.4 Conclusion and Research Gaps

This chapter depicts following conclusions

1. Coupling metrics in general, and CBO and RFC are specifically found useful in predicting software faults, irrespective of the dataset size, type of dependent variable.
2. Most of the coupling metrics discussed in Section 3.1.3, have not been addressed by the SFP community (see Table 3.5). This gap has been suggested in the future work.
3. Unused coupling metrics do not carry any exclusive coverage of coupling levels and principles.

4. Every time coupling metrics are evaluated individually. Hence, the combined impact of multiple coupling metrics in SFP needs to be evaluated.
5. Coupling metrics are evaluated using products developed in C, C++, Java, or COBOL. However, their performance in other extensively used programming languages, like Python, C#, and Visual basic are yet to be evaluated.

Likewise, few famous scripting languages like PHP, JavaScript, etc. may also be employed. This gap has been suggested in the future work.

6. Most of the metrics do not justify the complexity associated with the coupling levels. This thesis document focuses on this very aspect.

7. Content coupling, hiddenness, and rigidness are ignored by all the metrics. This gap has been suggested in the future work.

Chapter 4

Vovel Metrics: New Coupling Metrics

The coupling level is an important aspect that shows the degree of complexity between the coupled modules. The importance of this aspect is realized by many authors [24–27], while some others accommodate it in their proposed metrics also (see Table 3.5). Likewise, the data flow volume is another important dimension to reflect the nature of complexity between the coupled modules, which is also appreciated by many authors [118, 131, 133] and few coupling metrics accommodate these aspects also (see Table 3.5). The DataC metric covers one coupling level along with data flow volume[131]. However, both of these two aspects (i.e., coupling levels and volumd of information flow) have not been found in any of the metrics, therefore the impact of these two important factors on SFP is yet to be addressed. This **lack** can be addressed by first computing these two aspects using some metric (Section 4.1 and 4.2), and then evaluating the impact of the metric in SFP. Rest of this thesis document is dedicated to address these aspects.

4.1 Possible Solutions

This section analyses the possibilities to meet the objective by using existing coupling metrics. It is clear from Table 3.5 that individual metrics cannot achieve our objective. Yet, can a combination of multiple existing metrics address this issue? This section is dedicated to find the answer to this question.
We make six classes of the metrics based on the evaluation shown in Table 3.5.

These six classes and their corresponding metrics are shown in Table 4.1. Table

Table 4.1: Six classes of coupling metrics

Classes	**Levels' coverage**	**Discrimination of Levels**	**Volume**	**Metrics Included**
1	None	No	No	DAC, CDM
2	Partial	No	No	LIF, WTCoup, DC, Briand suite, ABC, Ca, DCC, Fan-in, CBO, RFC, OCM, CCMCOF, Fan-out, GIF, Ce, I, CFM
3	Partial	No	Yes	NIHICP, ICP
4	Partial	Yes	No	FM, OM, DM, OSMC
5	Full	No	No	C$\sim$, DCBO, CtrlC, MR, MPC, NAS, Nagappan suite, AM
6	Full	No	Yes	DataC, E, SC

4.1 is quite helpful to search for an answer to the posed question.

1. Combining multiple metrics belonging to only one class cannot solve the problem. The reason is that one class alone does not address all the required coupling aspects, even if all the metrics of that class are combined. For instance, Fan-in and Fan-out metrics belong to the same class. If we combine these two metrics, neither discrimination of levels nor volume would be covered.

2. Combining the metrics of Class 1 with any other class' metrics cannot solve the problem either. The reason is that in such a case, there would be a duplication of coupling aspects. For instance, by combining Fan-in for Class 1, and ICP of Class 3, discrimination of level cannot be covered. The same goes for combining metrics belonging to any two classes.

3. Combining metrics of Class 2, 3, or 4 with any metric of Class 5 or 6 will duplicate the coverage of some levels. And so, combining metrics of Class 5 with metrics of Class 6. Duplicate coverage of any factor will increase the value of the metric without the due importance of that very factor. For instance, Fan-in and SC belong to Class 2 and 5 respectively. Both of these metrics cover data coupling. If we combine them there would be the computing of data coupling twice, whereas such dual computation is not realistic.

4. Combining metrics of Class 2 with metrics of Class 3 is not useful because both classes' metrics do not address discrimination of coupling levels. For instance, by combining DCM and Fan-in discrimination of coupling level would be left unaddressed.

5. Combining metrics of Class 2 with metrics of Class 4 is also not useful, since both do not address data flow volume. Like, combining Fan-in and DM would leave the data flow volume unaddressed.

Keeping in view the above discussion only one possibility is left to be analyzed, i.e., combining Class 3 and 4 metrics. There are two metrics in Class 3, i.e., NIHICP and ICP. Both of these metrics are the same in coverage (see Table 3.5), therefore, we select only one metric, which is NIHICP. In Class 4, FM, OM, and OSMC are the same in converge therefore, we select only one of them, which is FM. Eventually, only two pairs suffice to be analyzed (NIHICP, DM) and (NIHICP, FM). In the first pair, Control, Stamp, and Data coupling levels are addressed twice, whereas, in the second pair, Control, Descriptive, Stamp, and data levels are addressed twice. Therefore, combining metrics of Class 3 and 4 cannot help to meet our objective.
The only possible solution left, is to derive a new metric that covers both of the aspects of coupling. Later, the metric may be used in SFP to evaluate the two aspects in SFP. The rest of this chapter is devoted to the derivation of such a metric.

4.2 Computation/Derivation of Vovel Metrics

Keeping in view the importance of data flow volume and coupling level, we propose two novel coupling metrics named; *Vovel-in* and *Vovel-out*. They both are collectively named as *Vovel* metrics. The name is composed out of the first two letters of word volume and the last three letters of level. This section elaborates the process of deriving/computing the proposed Vovel metrics. Figure 4.1 illustrates the components and composition of the proposed metrics.
The figure shows two steps for computing Vovel metrics, i.e., Computing data flow volume and coupling levels, which are elaborated in Section 4.2.1, and 4.2.2, respectively. Likewise, the terms shown in the figure, like $Vol(M)$, $v(M)$, etc., are also elaborated in the corresponding subsection.

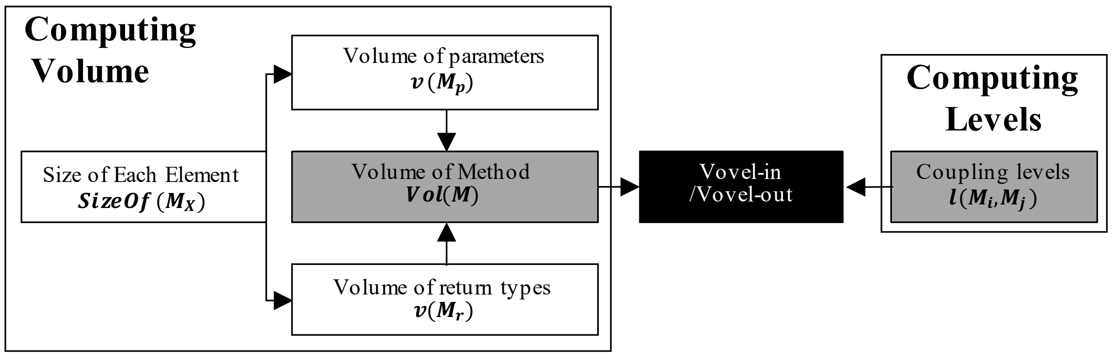

Computing Volume
Volume of parameters $v(M_p)$
Size of Each Element $SizeOf(M_X)$
Volume of Method $Vol(M)$
Volume of return types $v(M_r)$
Vovel-in /Vovel-out
Computing Levels
Coupling levels $l(M_i,M_j)$

FIGURE 4.1: Process of deriving Vovel metrics

4.2.1 Computing Data Flow Volume

Volume refers to the amount of data flow between modules, which is usually done through parameters and/or return values in the case of methods. The metrics like, GIF,LIF, DataC, SC, ICP, and NIHICP consider volume. Likewise, the dependency relationship covered by CSSM metric [185] considers the parameters and return types. However, these metrics consider only the number of parameters, whereas, the volume does not solely dependent on the number of parameters but the nature of parameters also. Like, primitive data types are relatively narrow in carrying information as compared to arrays. Therefore the volume address by these coupling metrics could not satisfy the due coverage of data flow volume. We compute the volume of method M, i.e., $Vol(M)$ using equation 4.1.

$$Vol(M) = \begin{cases} 1, & For\,content\,coupling \\ v(M_c), & For\,common\,coupling \\ v(M_p) + v(M_r), & \text{otherwise} \end{cases} \tag{4.1}$$

Where,

M_p is a list of parameters in method M, and $v(M_p)$ shows the volume of method M w.r.t. its parameters.

M_r is a list of return types in method M, and $v(M_r)$ shows the volume of method M w.r.t. its return types.

M_c is a list of common variable types that a method M reads or writes, and $v(M_c)$ shows the volume of the shared variables.

The $v(M_r)$ covers the languages that allow more than one value to be returned.

One such language is Python. All $v(M_p)$, $v(M_r)$, and $v(M_c)$ are computed by equation 4.2.

$$v(M_X) = \begin{cases} \sum_{j=1}^{n} WeightOf(M_{X_j}), & n > 0 \\ 1, & otherwise \end{cases} \tag{4.2}$$

whereas, in M_X, X can be p, r, and c to compute $v(M_p)$, $v(M_r)$, and $v(M_c)$, respectively. $WeightOf(M_{X_j})$ is the weight assigned to an element j from the list of parameter/return/common variable type X.

Whereas, in M_X, X can be p, r, and c to compute $v(M_p)$, $v(M_r)$, and $v(M_c)$, respectively. $WeightOf(M_{X_j})$ is the weight assigned to an element j from the list of parameter/return/common variable type X.

The weight would reflect the complexity associated with the datatype. Generally, there are two classes of datatype, i.e., primitive and composite. The primitive datatype includes, byte, short, int, long, float, double, boolean, and char. While composite datatype includes String, Arrays and Classes/Objects. Moreover, String is a collection of char. Whereas, Array can be collection of primitive datatypes and/or objects. Finally, Object may include all the types along with member functions. This concludes that if,

$$WeightOf(PrimitiveType) = w \tag{4.3}$$

$$and\ WeightOf(StringType) = x \tag{4.4}$$

$$and\ WeightOf(ArrayType) = y \tag{4.5}$$

$$and\ WeightOf(ObjectType) = z \tag{4.6}$$

Then,

$$w < x < y < z \tag{4.7}$$

Assigning the complexity associated with the datatypes can be an important contribution to work on. However, for the sake of simplicity and satisfying the condition mentioned in Equation 4.7, following conditional function has been used wherein weights are assigned as a heuristic.

$$WeightOf(P) = \begin{cases} 1, & P = PrimitiveType \\ 2, & P = StringType \\ 3, & P = ArrayType \\ 4, & P = ObjectType \end{cases} \tag{4.8}$$

4.2.2 Inducing Coupling Levels

Couplings can vary in complexity w.r.t. the levels [24–27]. The proposed metrics include ten such levels and assign weights using following function as per their complexity reported in the studies[24–27].
Whereas, $l(M_i, M_j)$ represents the coupling level l between any two methods shown by M_i and M_j. *No coupling* is assigned zero weight, which shows that there is not coupling between the modules, in fact only the control is being transferred from one module to another module. This level helps us to simplify the metrics' equation. The function $l(M_i, M_j)$ covers all the coupling levels discussed in Section 3.3.4 with the added segregation of stamp and scalar in common, control, and descriptive coupling. The reason is to accommodate the data flow volume at the coupling levels also since stamp coupling is broader than scalar coupling.

4.2.3 Relationship Between Vovel-in and Vovel-out

Keeping in view the mathematical forms shown in Equation 4.12 and 4.13, there are following similarities between them:

1. Both the metrics considers the volume of method
2. Both the metrics consider the level of coupling
3. Both the metrics compute levels of coupling with all the methods in a program irrespective of the presence or absence coupling.

Likewise, there exist following difference between them

1. Vovel-in computes inflow to a module, whereas Vovel-out computes the outflow from the a module.
2. Vovel-in computes the volume of calling module, whereas Vovel-out computes the volume of called module
3. In Vovel-in, volume of module is multiplied to the sum of all levels of coupling between calling module and remaining modules. Whereas in Vovel-out,

volume of called module is multiplied to only the level of coupling exists between calling and called module.

$$l(M_i, M_j) = \begin{cases} 0, & No\,coupling\,from\,M_i\,to\,M_j \\ 1, & Call\,coupling\,from\,M_i\,to\,M_j \\ 2, & Data\,coupling\,from\,M_i\,to\,M_j \\ 3, & Stamp\,coupling\,from\,M_i\,to\,M_j \\ 4, & Scalar\,descriptive\,coupling\,from\,M_i\,to\,M_j \\ 5, & Stamp\,descriptive\,coupling\,from\,M_i\,to\,M_j \\ 6, & Scalar\,control\,from\,M_i\,to\,M_j \\ 7, & Stamp\,control\,from\,M_i\,to\,M_j \\ 8, & Scalar\,common\,between\,M_i\,and\,M_j \\ 9, & Stamp\,common\,between\,M_i\,and\,M_j \\ 10, & Content\,coupling\,from\,M_i\,and\,M_j \end{cases} \tag{4.9}$$

Since, the inflow and outflow are not necessary to have linear relationship, Vovel metrics would not carry any linear relationship. Likewise, the volume and usage of modules are independent of eachother, this further adds to non-linearity between the two metrics.

4.2.4 Combining Coupling Levels and Data Flow Volume

Since, the coupling levels are directional (except common coupling), we derive two metrics *Vovel-in* and *Vovel-out* to accommodate two distinct direction. These two metrics are computed by combining the function $l(M_i, M_j)$ and equation 4.1. The *Vovel-in* and *Vovel-out* of a method M can be computed by equation 4.10 and 4.11, respectively.

$$Vovel\text{-}in(M) = \sum_{j=1}^{m} l(M_j, M) \times Vol(M) = Vol(M) \times \sum_{j=1}^{m} l(M_j, M) \tag{4.10}$$

$$Vovel\text{-}out(M) = \sum_{j=1}^{m} l(M, M_j) \times Vol(M_j) \tag{4.11}$$

Where, m is the number of all the methods in the software product excluding M. The equations compute coupling of a **method** with other methods. However,

TABLE 4.2: Coverage of coupling levels and principles by Vovel metrics

Metric	**Levels**							**Principles**		
	Content	**Common**	**Control**	**Descriptive**	**Stamp**	**Data**	**Call**	**Volume**	**Hidden**	**Rigid**
Vovel-in	●	●	●	●	●	●	●	✓	✗	✗
Vovel-out	●	●	●	●	●	●	●	✓	✗	✗

the equations can slightly be modified to equation 4.12 and 4.13 to compute the coupling of a **class** with other classes.

$$Vovel\text{-}in(C) = \sum_{i=1}^{n}\sum_{j=1}^{m} l(M_j, M_i) \times Vol(M_i) \tag{4.12}$$

$$Vovel\text{-}out(C) = \sum_{i=1}^{n}\sum_{j=1}^{m} l(M_i, M_j) \times Vol(M_j) \tag{4.13}$$

Where, n is a number of methods in class C and m is the number of all the methods belonging to other classes. In the equations 4.10, 4.11, 4.12, and 4.13 volume of a method being called is computed.

The product in the Vovel metrics has an important implication. In Vovel-in, whenever the module is called same volume of information would be flowed every ***time***. This requires the volume of information of called module to be multiplied by "the way it is being called" (i.e., level of coupling). Likewise, in Vovel-out, since there would be multiple modules being called. Therefore, volume of information of the corresponding module would be flowed every ***time***. This requires the volume of information associated with that very module should be multiplied by "the way it is being called" (i.e., level of coupling).

4.3 Significance of Vovel Metrics

The proposed metrics cover all the important aspects which are used to evaluate other metrics (see Table 4.2). In addition to that, the proposed metrics have some unique significance also,

1. The metrics accommodate both structural and OO paradigm.

2. Some programming languages do not support multiple values to be returned, while some others do. However, the metric supports both types of languages.

3. Table 3.4 shows differences in the number of coupling levels and the placement of coupling levels. These differences can be accommodated by just modifying the function $l(M_i, M_j)$. Hence, the proposed metrics are adaptive enough to accommodate the difference in numbers of coupling levels and diversity of coupling levels' placement.

4.4 Examples

In this section we demonstrate the computation of Vovel metrics which are derived in Section 4.2. The Vovel metrics work for both structural and OO paradigm. First part of the computation comprises the computation of volume which is paradigm independent is demonstrated in Section 4.4.1. Second part of the computation comprises the coupling between methods which is paradigm dependent. Therefore we demonstrate the metrics computation for Structural and OO paradigm saperately in Sections 4.4.2 and 4.4.3 respectively.

TABLE 4.3: Hypothetical Java based methods and their volume

Coupled component	$v(M_r)$	$v(M_p)$	$v(M_c)$	$Vol(M)$
`void` **`A`**`()`	0	0	-	0
`void` **`B`**`(int)`	0	1	-	1
`void` **`C`**`(boolean, short)`	0	1+1	-	2
`void` **`D`**`(float, char, bool)`	0	1+1+1	-	3
`int` **`E`**`()`	1	0	-	1
`char` **`F`**`(boolean)`	1	1	-	2
`boolean` **`G`**`(double, int[])`	1	1+3	-	5
`int` **`H`**`(int, String, object)`	1	1+2+4	-	8
`int` **`C`**	-	-	1	1
`void` **`X()`**		0	-	0

4.4.1 Computing Volume of Methods

Volume refers to the amount of data flow between modules, which is usually done through parameters and/or return values in case of methods. We use equations 4.1

and 4.2 for the computation of volume. The equation 4.2 considers the memory allocated to the data type, which is dependent on the programming language. We use Java for the following examples.
In Table 4.3 we assign 16 bit size to the object since it is the minimum object size for modern 64-bit JDK object. However, in reality, we consider the memory allocated to an object, which is implementation-dependent, so it may be equal to or greater than 16. Finally boxed types, arrays, Strings and other containers like multidimensional arrays, memory allocated is implementation-dependent. In Java one way to get an estimate of these container sizes is to implement Instrumentation interface.

4.4.2 Computing Vovel Metrics in Structure Paradigm

In the following examples, methods are denoted by a circle with the name inside it. The arrow is directed from the caller to called method. The label on the arrow shows the type of coupling between the methods on either sides of an arrow.

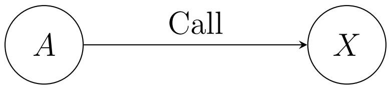
4.4.2.1 Example 1

$$
\begin{aligned}
Vovel\text{-}in(A) &= Vol(A) \times l(B, A) \\
&= 0 \times 0 \\
&= 0 \\
Vovel\text{-}in(B) &= Vol(B) \times l(A, B) \\
&= 1 \times 0 \\
&= 0 \\
Vovel\text{-}out(A) &= Vol(B) \times l(A, B) \\
&= 1 \times 0 \\
&= 0 \\
Vovel\text{-}out(B) &= Vol(A) \times l(B, A) \\
&= 0 \times 0 \\
&= 0
\end{aligned}
$$

4.4.2.2 Example 2

$$\begin{aligned}
Vovel\text{-}in(C) &= Vol(C) \times l(D,C)\\
&= 2 \times 0\\
&= 0\\
Vovel\text{-}in(D) &= Vol(D) \times l(C,D)\\
&= 3 \times 0\\
&= 0\\
Vovel\text{-}out(C) &= Vol(D) \times l(C,D)\\
&= 3 \times 0\\
&= 0\\
Vovel\text{-}out(D) &= Vol(C) \times l(D,C)\\
&= 2 \times 0\\
&= 0
\end{aligned}$$

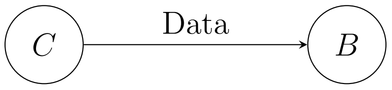
4.4.2.3 Example 3



$$\begin{aligned}
Vovel\text{-}in(A) &= Vol(X) \times l(X,A)\\
&= 0 \times 0\\
&= 0\\
Vovel\text{-}in(X) &= Vol(A) \times l(A,X)\\
&= 0 \times 1\\
&= 0
\end{aligned}$$

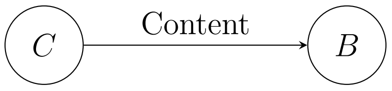
$$\begin{aligned}
Vovel\text{-}out(A) &= Vol(A) \times l(A, X)\\
&= 0 \times 0\\
&= 0\\
Vovel\text{-}out(X) &= Vol(X) \times l(X, A)\\
&= 0 \times 0\\
&= 0
\end{aligned}$$

4.4.2.4 Example 4



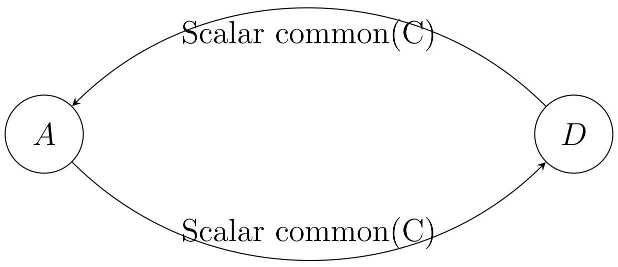
$$\begin{aligned}
Vovel\text{-}in(B) &= Vol(B) \times l(C, B)\\
&= 1 \times 2\\
&= 2\\
Vovel\text{-}in(C) &= Vol(C) \times l(B, C)\\
&= 2 \times 0\\
&= 0\\
Vovel\text{-}out(B) &= Vol(C) \times l(B, C)\\
&= 2 \times 0\\
&= 0\\
Vovel\text{-}out(C) &= Vol(B) \times l(C, B)\\
&= 1 \times 2\\
&= 2
\end{aligned}$$

4.4.2.5 Example 5



$$\begin{aligned}
Vovel\text{-}in(B) &= Vol(B) \times l(C,B)\\
&= 1 \times 1 = 1\\
Vovel\text{-}in(C) &= Vol(C) \times l(B,C)\\
&= 2 \times 0 = 0\\
Vovel\text{-}out(B) &= Vol(C) \times l(B,C)\\
&= 2 \times 0 = 0\\
Vovel\text{-}out(C) &= Vol(B) \times l(C,B)\\
&= 1 \times 10 = 10
\end{aligned}$$

4.4.2.6 Example 6



$$\begin{aligned}
Vovel\text{-}in(A) &= Vol(A) \times l(D,A)\\
&= 0 \times 8\\
&= 0\\
Vovel\text{-}in(D) &= Vol(D) \times l(A,D)\\
&= 3 \times 8\\
&= 24\\
Vovel\text{-}out(A) &= Vol(D) \times l(A,D)\\
&= 3 \times 8\\
&= 24\\
Vovel\text{-}out(D) &= Vol(A) \times l(D,A)\\
&= 0 \times 8\\
&= 0
\end{aligned}$$

4.4.2.7 Example 7

$$\begin{tikzcd}
C & D \arrow[l, "\text{Data}"'] \arrow[r, "\text{Scalar Descriptive}"] & G
\end{tikzcd}$$

$$\begin{aligned}
Vovel\text{-}in(G) &= Vol(G) \times l(C,G) + Vol(G) \times l(D,G) \\
&= 5 \times 0 + 5 \times 4 \\
&= 20 \\
Vovel\text{-}in(C) &= Vol(C) \times l(G,C) + Vol(C) \times l(D,C) \\
&= 2 \times 0 + 2 \times 2 \\
&= 4 \\
Vovel\text{-}in(D) &= Vol(D) \times l(G,D) + Vol(D) \times l(C,D) \\
&= 3 \times 0 + 3 \times 0 \\
&= 0 \\
Vovel\text{-}out(G) &= Vol(C) \times l(G,C) + Vol(D) \times l(G,D) \\
&= 2 \times 0 + 3 \times 0 \\
&= 0 \\
Vovel\text{-}out(C) &= Vol(G) \times l(C,G) + Vol(D) \times l(C,D) \\
&= 5 \times 0 + 3 \times 0 \\
&= 0 \\
Vovel\text{-}out(D) &= Vol(G) \times l(D,G) + Vol(C) \times l(D,C) \\
&= 5 \times 4 + 2 \times 2 \\
&= 24
\end{aligned}$$

4.4.2.8 Example 8

$$\begin{tikzcd}
E & F \arrow[r, "\text{Scalar control}"] & H
\end{tikzcd}$$

$$\begin{aligned}
Vovel\text{-}in(E) &= Vol(E) \times l(F,E) + Vol(E) \times l(H,E) \\
&= 1 \times 0 + 1 \times 0 \\
&= 0 \\
Vovel\text{-}in(F) &= Vol(F) \times l(E,F) + Vol(F) \times l(H,F) \\
&= 2 \times 0 + 2 \times 0 \\
&= 0 \\
Vovel\text{-}in(H) &= Vol(H) \times l(E,H) + Vol(H) \times l(F,H) \\
&= 8 \times 0 + 8 \times 6 \\
&= 48 \\
Vovel\text{-}out(E) &= Vol(F) \times l(E,F) + Vol(H) \times l(E,H) \\
&= 2 \times 0 + 8 \times 0 \\
&= 0 \\
Vovel\text{-}out(F) &= Vol(E) \times l(F,E) + Vol(H) \times l(F,H) \\
&= 1 \times 0 + 8 \times 6 \\
&= 48 \\
Vovel\text{-}out(H) &= Vol(E) \times l(H,E) + Vol(F) \times l(H,F) \\
&= 1 \times 0 + 2 \times 0 \\
&= 0
\end{aligned}$$

4.4.3 Computing Vovel Metrics at Class Levels

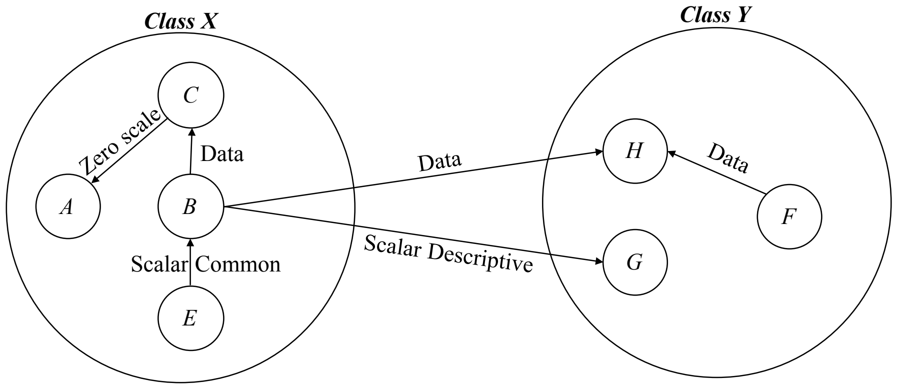


Figure 4.2: Example for computing Vovel metrics at class levels

4.4.3.1 Computing Vovel-in metric for Class X

$$\begin{aligned}
Vovel\text{-}in(A) &= Vol(A) \times l(F,A) + Vol(A) \times l(G,A) + Vol(A) \times l(H,A)\\
&= 0 \times 0 + 0 \times 0 + 0 \times 0\\
&= 0\\
Vovel\text{-}in(C) &= Vol(C) \times l(F,C) + Vol(C) \times l(G,C) + Vol(C) \times l(H,C)\\
&= 2 \times 0 + 2 \times 0 + 2 \times 0\\
&= 0\\
Vovel\text{-}in(B) &= Vol(B) \times l(F,B) + Vol(B) \times l(G,B) + Vol(B) \times l(H,B)\\
&= 1 \times 0 + 1 \times 0 + 1 \times 0\\
&= 0\\
Vovel\text{-}in(E) &= Vol(E) \times l(F,E) + Vol(E) \times l(G,E) + Vol(E) \times l(H,E)\\
&= 1 \times 0 + 1 \times 0 + 1 \times 0\\
&= 0\\
Vovel\text{-}in(X) &= Vovel\text{-}in(A) + Vovel\text{-}in(C) + Vovel\text{-}in(B) + Vovel\text{-}in(E)\\
&= 0 + 0 + 0 + 0 = 0
\end{aligned}$$

4.4.3.2 Computing Vovel-out Metric for Class X

$$\begin{aligned}
Vovel\text{-}out(A) &= Vol(F) \times l(A,F) + Vol(G) \times l(A,G) + Vol(H) \times l(A,H)\\
&= 2 \times 0 + 5 \times 0 + 8 \times 0\\
&= 0\\
Vovel\text{-}out(C) &= Vol(F) \times l(C,F) + Vol(G) \times l(C,G) + Vol(H) \times l(C,H)\\
&= 2 \times 0 + 5 \times 0 + 8 \times 0\\
&= 0\\
Vovel\text{-}out(B) &= Vol(F) \times l(B,F) + Vol(G) \times l(B,G) + Vol(H) \times l(B,H)\\
&= 2 \times 0 + 5 \times 4 + 8 \times 2\\
&= 36\\
Vovel\text{-}out(E) &= Vol(F) \times l(E,F) + Vol(G) \times l(E,G) + Vol(H) \times l(E,H)\\
&= 2 \times 0 + 5 \times 0 + 8 \times 0\\
&= 0
\end{aligned}$$

$$\begin{aligned}
Vovel\text{-}out(X) &= Vovel\text{-}out(A) + Vovel\text{-}out(C) + Vovel\text{-}out(B) + Vovel\text{-}out(E)\\
&= 0 + 0 + 36 + 0\\
&= 36
\end{aligned}$$

4.4.3.3 Computing Vovel-in metric for Class Y

$$\begin{aligned}
Vovel\text{-}in(F) &= Vol(F) \times l(A, F) + Vol(F) \times l(B, F)\\
&+ Vol(F) \times l(C, F) + Vol(F) \times l(E, F)\\
&= 2 \times 0 + 2 \times 0 + 2 \times 0 + 2 \times 0\\
&= 0\\
Vovel\text{-}in(G) &= Vol(G) \times l(A, G) + Vol(G) \times l(B, G)\\
&+ Vol(G) \times l(C, G) + Vol(G) \times l(E, G)\\
&= 5 \times 0 + 5 \times 4 + 5 \times 0 + 5 \times 0\\
&= 20\\
Vovel\text{-}in(H) &= Vol(H) \times l(A, H) + Vol(H) \times l(B, H)\\
&+ Vol(H) \times l(C, H) + Vol(H) \times l(E, H)\\
&= 8 \times 0 + 8 \times 2 + 8 \times 0 + 8 \times 0\\
&= 16\\
Vovel\text{-}in(Y) &= Vovel\text{-}in(F) + Vovel\text{-}in(G) + Vovel\text{-}in(H)\\
&= 0 + 20 + 16 = 36
\end{aligned}$$

4.4.3.4 Computing Vovel-out metric for Class Y

$$\begin{aligned}
Vovel\text{-}out(F) &= Vol(A) \times l(F, A) + Vol(B) \times l(F, B)\\
&+ Vol(C) \times l(F, C) + Vol(E) \times l(F, E)\\
&= 0 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0\\
&= 0\\
Vovel\text{-}out(G) &= Vol(A) \times l(G, A) + Vol(B) \times l(G, B)\\
&+ Vol(C) \times l(G, C) + Vol(E) \times l(G, E)\\
&= 0 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0\\
&= 0
\end{aligned}$$

$$\begin{aligned}
Vovel\text{-}out(H) &= Vol(A) \times l(H, A) + Vol(B) \times l(H, B) \\
&+ Vol(C) \times l(H, C) + Vol(E) \times l(H, E) \\
&= 0 \times 0 + 1 \times 0 + 1 \times 0 + 1 \times 0 \\
&= 0 \\
Vovel\text{-}out(Y) &= Vovel\text{-}out(F) + Vovel\text{-}out(G) + Vovel\text{-}out(H) \\
&= 0 + 0 + 0 \\
&= 0
\end{aligned}$$

Chapter 5

Materials and Methods

The usefulness of the *Vovel* metrics in SFP using ML is subject to be validated empirically. Such validation requires three components; dataset, modeling technique, and performance measures. This chapter is dedicated to the selection and brief elaboration on these three components.

5.1 Datasets Development

The availability of datasets is a base requirement for any ML base activity. One possible solution is to use public datasets. Section 2.1.2 elaborates the list of public datasets. These datasets lack the information of *Vovel* metrics, nor have any metric the *Vovel* metric may be derived from. Another solution is to build new datasets by parsing the required metrics. In this section, we discuss three steps to build the datasets. The first step is the selection of software projects which is elaborated in subsection 5.1.1. The second step is the parsing of the required metrics from the selected projects, which is discussed in subsection 5.1.2. Third, the final step is to add fault information to the instances of the datasets, which is elaborated in subsection 5.1.3. The outcome of this phase is the developed five datasets. The statistical description of these datasets is shown in Table 5.1.

5.1.1 Case Study

The proposed metrics need to be validated empirically for viability. D'Ambros *et al.* [142] develop fault datasets of five projects; **A**pache Lucene 2.4[1], **E**clipse

[1]Lucene.apache.org

Equinox Framework 3.4[2], **E**clipse JDT Core 3.4[3], **E**clipse PDE UI 3.4.1[4], and **M**ylyn 3.1[5] having 691, 439, 997, 1562, and 2196 classes respectively. These datasets are shortly named as AEEEM. The rational of selecting these datasets are as following:

1. The source code of the selected dataset is publicly available. Therefore, we can parse the relevant information, more specifically the Vovel metrics out of it.
2. The selected datasets have information of 61 metrics, including the five well-known coupling metrics, i.e. Ce, CBO, RFC, Fan-in, and Fan-out.
3. The selected datasets are developed in Java and there are numerous of freely available parsers that can parse information from Java code. Some of these parsers are, AEA [219], AMT[220], Together[221], ckjm[222], Viewer[223], JCAT[224], JCTIViz[225], JMCT[226], JMT, JCMT, Understand[227], etc.
4. Frequent usage of AEEEM in SFP [53, 209, 228].

A brief description of these projects is as follows.

Apache Lucene is a high-performance, full-featured text search engine library written entirely in Java. It is a technology suitable for nearly any application that requires full-text search, especially cross-platform. Apache Lucene is an open-source project available for free download.

Eclipse equinox is an implementation of the OSGi core framework specification, a set of bundles that implement various optional OSGi services and other infrastructure for running OSGi-based systems. It is responsible for developing and delivering the OSGi framework implementation used for all of Eclipse. The Equinox OSGi core framework implementation is used as the reference implementation. The goal of the Equinox project is to be a first-class OSGi community and foster the vision of Eclipse as a landscape of bundles.

Eclipse JDT Core is a Java infrastructure of the Java IDE. It includes an

[2]www.eclipse.org/equinox/
[3]www.eclipse.org/jdt/core/
[4]www.eclipse.org/pde/pde-ui/
[5]www.eclipse.org/mylyn/

incremental Java compiler. In particular, it allows to run and debug code that still contains unresolved errors. It provides a Java-centric view of a project. It also carries a Java document model providing API for manipulating a structured Java source document.

Eclipse PDE UI provides a comprehensive set of tools to create, develop, test, debug and deploy Eclipse plug-ins. PDE UI also provides multi-page editors that centrally manage all manifest files of a plug-in or feature. It carries new project creation wizards to create a new plug-in, fragment, feature, feature patch, and update sites.

Mylyn is the task and application life cycle management framework for Eclipse. It provides a revolutionary task-focused interface and a task management tool for developers.

5.1.2 Parsing of Metrics Information

D'Ambros *et al.* [142] compute numerous software product metrics from the selected five projects. Out of these metrics, we selected five coupling metrics, i.e, Ce, CBO, Fan-in, Fan-out, and RFC for the following reasons:

1. These coupling metrics provide coverage to most of the coupling levels that are discussed earlier (as shown in Table 3.5).

2. The information of these metrics in the selected case studies is publicly available. That makes it easier for the researchers to redo the experiments performed in Chapter 6.

3. These coupling metrics are reported effective in SFP by numerous studies (see Section 3.2).

4. These coupling metrics are the most frequently used coupling metrics in SFP, as shown in Table 3.3.

In addition to the conventional coupling metrics, we computed the Vovel metrics (i.e. Vovel-in, Vovel-out)using Javaparser. Javaparser contains a set of libraries implementing a Java 1.0 to analyze and parse the Java projects. Its libraries provide an Abstract Syntax Tree (AST) of Java code. The AST structure then allows working with Java code in an easy programmatic way. It is used in various studies also [229–231]. The statistical description of the metrics in all five datasets is shown in Table 5.1.

TABLE 5.1: Statistical description of metrics in the selected datasets

Datasets	**Params.**	**Ce**	**CBO**	**RFC**	**Fan-in**	**Fan-out**	**Vovel-in**	**Vovel-out**
Apache Lucene 2.4	count	491	491	491	491	491	491	491
	mean	5.4	6.9	18.5	4.4	5.5	344.1	547.6
	std	5.4	7.6	23.4	12.1	6.8	2003	2983
	min	0	0	1	0	0	0	0
	25%	1	2	6	1	2	0	21
	50%	6	4	12	1	3	0	129
	75%	7	9	23	4	7	87	423
	max	81	64	308	174	67	57776	84187
Eclipse equinox framework 3.4	count	439	439	439	439	439	439	439
	mean	10.3	6.6	19.8	3.4	8.4	544.2	945.7
	std	9	8.3	27.9	5	10	3034	5034.5
	min	0	0	1	0	0	0	0
	25%	2.7	1.5	5	1	2	0	0
	50%	10	5	11	2	5	0	131
	75%	15	8	23	4	11	158	678
	max	105	56	213	32	67	57776	84187
Eclipse JDT Core 3.4	count	997	997	997	997	997	997	997
	mean	12	14.5	37	5.4	7.4	503.6	413.3
	std	17	19.4	55.9	13.7	9.7	2259.1	566.9
	min	0	0	0	0	0	0	0
	25%	2	4	9	1	2	0	86.8
	50%	7	9	20	2	4	25.8	296.3
	75%	20	18	42	4	10	243	539
	max	300	214	600	137	93	30181.3	9041.5

Eclipse PDE UI 3.4.1	count	1562	1562	1562	1562	1562	1562	1562
	mean	5.3	6.6	16.9	4.1	5.8	348.4	416.2
	std	5.2	7.6	20.7	13.4	6.8	1846.5	1710.3
	min	0	0	1	0	0	0	0
	25%	4	2	5	1	1	0	21
	50%	6	4	10	1	4	0	129
	75%	7	9	21	3	8	84.5	400.5
	max	110	80	308	355	67	57776	84187
Mylyn 3.1	count	2196	2196	2196	2196	2196	2196	2196
	mean	9	6.1	16.2	4.4	5.2	315.5	431.6
	std	6.4	7.2	21	13.8	6.5	1694.2	2091.8
	min	0	0	1	0	0	0	0
	25%	6	1	5	1	1	0	18
	50%	7	4	10	1	3	0	119
	75%	9	8	20	3	7	70	374
	max	94	80	308	223	67	57776	84187

5.1.3 Fault Labeling

The dependent variables that we used in our study are *fp* and *nfp*. The labeling is performed by D'Ambros *et al.* [142]. The authors used change log information, source code version information, and defect information linked to classes for both the prediction and validation. Finally, a defect tracking system (Bugzilla/Jira) is used for labeling. We rely on their labels. However, we convert the numerical bug label to binary variable by converting 0 bugs to *nfp*, and *fp* otherwise. Figure 5.1 shows the fault ratio in the selected projects.

5.2 ML Algorithm/Techniques

As discussed in Section 2.1.3, in SFP studies ML is the most dominating in terms of usage and performance. It has been used in 66% of papers published after 2005 [16]. This section is dedicated to eloborate the selected ML techniques which are later used for the evaluation of the proposed metric.

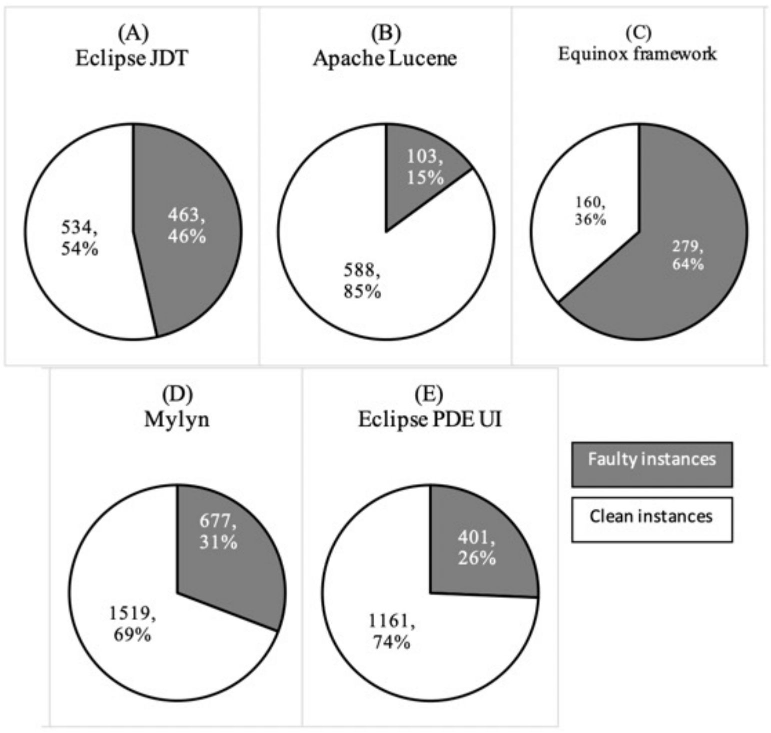


FIGURE 5.1: Ratio of faulty and clean instances in the selected datasets

5.2.1 Univariate Logistic Regression

Logistic regression is a standard statistical technique based on maximum likelihood estimation [232]. The logistic regression model expresses the relationship between Y_i and X_i in term of the conditional probability $P(Y_i = 1|X_i)$, as:

$$\log(\frac{P(Y_i = 1|X_i)}{1 - P(Y_1 = 1|X_i)}) \tag{5.1}$$

Where X is computed by the following equation:

$$\pi(X) = \frac{e^{C_0+C_1X}}{1 + e^{C_0+C_1X}} \tag{5.2}$$

where X is a set of independent variables which is coupling metrics in our case and p is the probability of occurrence of a fault in a class, which is a dependent variable. We aim to use ULR, for each coupling metric, against the probability of occurrence of a fault and determine if the measure is statistically related to a fault proneness.
The reason for the selection of Logistic regression models (ULR and later MLR) is because of the following reasons

1. LR models are relatively more resilient to overfitting than that of Multilayer

perceptron (MLP)

2. LR models are easier to develop and reproduce than that of MLP for having less number of hyperparameter optimization

3. LR models are faster in prediction at the deployment environment than that of the Decision tree, Random Forest, MLP

4. In LR there is no requirement of discretization as that of in Decision Tree, Random Forest, etc.

5. Hall *el al.* reported LR models as the most effective in SFP [15].

6. The survey of SFP studies from 1995 to 2018 conducted by Son *et al.* LR is amongst the top three used algorithms in SFP [138].

7. The metrics that we include in our experiment are positively oriented metrics. It means higher values are discouraged. That makes the data is linearly separable and thus makes the LR model a better candidate for modeling.

8. The selected projects were too small to be investigated with more sophisticated models.

9. According to Lessmann *et al.* simple algorithms (like LR) are not significantly worse than the sophisticated data processing techniques [159]

10. Numerous studies [19, 33, 35, 38, 39, 39, 40, 47, 54, 209] in SFP using coupling metrics also utilized LR.

5.2.2 Correlation Analysis

Correlation analysis shows the linear association between two metrics. In general, an absolute correlation coefficient greater than 0.8 between two metrics indicates the existence of a strong association. It is performed using either Pearson correlation or Spearman correlation. In both techniques correlation coefficient (r) ranges from -1 to 1, where -1 denotes a strong negative correlation and 1 denotes a strong positive correlation.

Strong correlation indicates the presence of redundant/duplicate information in two metrics. This duplicate/redundant information leads to model overfitting.

Hence, the resultant model will be less precise and less generalized.
In the case of a strong correlation between two metrics, one is generally ignored during model development. The removal of highly correlated factors should be done manually instead of using automatic techniques, such as stepwise variable selection because they may remove metric of interest in favor of less important metric.
In this thesis document, we used Spearman correlation because it can visualize the monotonic relationship (whether linear or not) between variables. While Pearson can only capture the linear relationship. Spearman correlation coefficient (r) between two variables is computed by the following formula:

$$r = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \tag{5.3}$$

where d is the distance between two metrics at the instance level and n is the total number of observations.

5.2.3 Multivariate Logistic Regression

Multivariate logistic regression (MLR) is an extension of ULR. It is used where more than one metrics are to be analyzed for their effect on predicting fault-prone components. In our experiment, we aim to construct MLR for the best fitting model to describe the relationship between dependent and independent variables. The outcome of the MLR is a fitted logistic regression equation. The MLR is computed through the following equation.

$$log\frac{\pi(X)}{\pi X - 1} = C_0 + C_1X_1 + C_2X_2 + ... + C_nX_n \tag{5.4}$$

where C_i is the coefficient associated with each independent variable.

5.3 Selection of Performance Measure

The performance of a model (ULR or MLR) is dependent on the performance measure [233] or sometimes multiple performance measures [234]. Therefore, the selection of performance measures while keeping in view their scope, relationship and interpretation is of key importance. The task is even more important when two performance measures consent on two classifiers' performance on one test set, which may conflict with some other test set.

In the classification, performance measures are derived out of a confusion matrix, which is a useful tool for analyzing the goodness of a classifier. The confusion matrix has four cases; true positive (TP), false negative (FN), true negative (TN), and false-positive (FP). If a classifier declares any faulty instance as faulty, the classification is *TP*. If a classifier declares any instance as fault-free when it is faulty, the classification is *FN*. If the classifier declares any fault-free instance as fault-free, the classification is *TN*. Finally, if a classifier labels any instance as faulty when it is fault-free, the classification is *FP*. Total positive instances are denoted by P and total negative instances are denoted by N.

The performance measures which are commonly used in SFP studies can be classified as either Positive and negative oriented measures. Positively oriented measures refer to the performance measure whose higher value is desirable. Among the existing classifiers' performance measures, Precision, TPR, TNR, Accuracy, F-measure, G-mean1, G-mean2, and J coefficient are positively oriented measures. In contrast to that, where lower values are desirable, are referred to as negatively oriented measures. FPR, FNR, Type-I error, Type-II error, Error rate, and Balance are all negatively oriented measures.

5.3.1 Evaluation of Performance Measures

We evaluated 14 performance measures through three evaluation measures; Plausibility, Consistency, and Discriminancy. The following subsections briefly describe the whole process.

5.3.1.1 Plausibility

Plausibility refers to the absence of implausible values. Implausible values can be of three types:

1. Occurrence of "divided by zero". For instance, for (0, 0, 1, 0). F-measure will produce "divided by zero" error. Thus, F-measure will be declared as an implausible value carrier in this particular scenario.
2. Non-minimum value of positively oriented measures or non-maximum value of negatively oriented measures for the worst classification. The worst clas-

sification occurs when none of the instances is correctly classified. One such scenario is (0, 1, 0, 0), wherein Type-I error is 0.5, which is non-maximum value. Thus, Type-I error will be declared as an implausible value carrier in this scenario.

3. Non-maximum value of positively oriented measures or non-minimum value of negatively oriented measures for the best classification. The best classification is achieved when all the instances are correctly classified. One such scenario is (1, 0, 1, 0).

The second and third types are in a mutually exclusive relationship, while they both can have co-occurrence with the first type. This evaluation parameter is named "plausibility" because implausible value carrier measure either provides insufficient information or does not provide any information about the goodness of a classifier.

5.3.1.2 Consistency and Discriminancy

Consistency and discriminancy between performance measures are proposed in [235]. Later, S. Rosset [236] computes discriminancy between AuC and Error rate, and Haung *et al.* [237] use discriminancy and consistency to evaluate AuC and Accuracy.
Consistency computes the consensus of two performance measures on evaluating different classifiers. It is a symmetric relationship. Two performance measures, say f and g are consistent with each other when comparing two algorithms a and b, if both f and g stipulate that a is better than b. However, if f stipulates that a is better than b and g contradicts that, then f and g are said to be inconsistent.
Discriminancy is an ability of one performance measure over another to discriminate different classifiers. For instance, if f declares a as different (better or worse) than b, while g declares both a and b as equivalent. Then f is more discriminating that g.

5.3.1.3 Evaluation Result

In plausibility analysis, the base measures are identified as the worst of all the measures except Precision. F-measure and G-mean1 are the best of all measures.

However, if there is a chance of occurrence of zero which needs to be considered manually.
Further, we conclude that F-measure and G-mean1 are equally the most discriminating performance measures, hence the most suitable to evaluate two different classifiers. Besides this, both of these measures are found to be the least scorer in plausibility analysis. So, it is good to use one of them when there is NO possibility of occurrence of zero in any of the four cases. However, where there is a chance of occurrence of zero in any of the cases, Precision would be relatively a good choice for having more plausibility scores and more discriminating.
Keeping in view the above justification and our selected datasets, we choose F-measure as a performance measure.

5.3.2 Selection of β Value in F-measure

F-measure is the harmonic mean between Precision and TPR, first introduced in [164]. It tells how precise a classifier is (how many instances it classifies correctly), as well as how robust it is. Mathematically it is written as.

$$F\text{-}measure = \frac{(\beta^2 + 1) \times Precision \times TPR}{\beta^2 \times Precision + TPR} \tag{5.5}$$

Where, β can be 0.5, 1, and 2 for F0.5, F1, and F2 measures respectively. Varying the value of β allows different weights to be assigned to false negative and false positive. Detail derivation of F-measure and assigning the value of β are discussed in [164].
In our work, a value of $\beta = 1$ has been used, that equalizes the influence of false negatives and false positives. Since declaring *fp* module as *nfp* has a performance overhead. Whereas, declaring a *nfp* module as *fp* incurs greater testing cost. We need to have a balance between false negative and false positive. Therefore, F1 score is more meaningful than the F0.5 and F2 score in the current context.

5.4 Statistical Significance Assessment

Models are commonly evaluated using resampling methods like k-fold cross-validation. This approach can be misleading as it is hard to know whether the difference between mean skill scores is real or the result of a statistical fluke. To address this

problem, statistical significance tests are designed to assess the generalization and significance of the reported result.
The statistical significance test needs some hypotheses to assess and the significance test. Both of these aspects are discussed below.

5.4.1 Formulation of Hypotheses

Keeping in view our objective we construct the following hypotheses.
H_0 : The data flow volume and coupling levels do not improve the performance of SFP when used in combination with existing coupling metrics.
H_1**:** The data flow volume and coupling levels improve the performance of SFP when used in combination with existing coupling metrics.
First, we will test the null hypothesis H_0 if it is rejected, then the alternative hypothesis H_1 will be tested. Otherwise, the null hypothesis will be accepted and the alternative hypothesis will be rejected.
Since, the data flow volume and coupling levels are computed in Vovel metrics, we further refine our hyptheses. We made two sets of features that act as independent variables. These sets and their corresponding elements are as follows:
$Set_1 = \{Ce, CBO, Fan\text{-}in, Fan\text{-}out, RFC\}$
$Set_2 = Set_1 \cup \{Vovel\text{-}in, Vovel\text{-}out\}$
The above hypotheses can further be stated as;
H_0 : The Set_2 is as effective in SFP as Set_1.
H_1 : The Set_2 is more effective in SFP than that of Set_1.

5.4.2 Selection of Statistical Test

For statistical testing, we used Wilcoxon signed-rank test proposed by Frank Wilcoxon[238]. Wilcoxon signed-rank test is a nonparametric test for pre-and post-treatment measurements. In our case, it is with and without inducing Vovel metrics. It does not assume the normal distribution of the samples. Wilcoxon signed-rank test incorporates more information about the data, it is more powerful than the sign test. The test has three assumptions.

1. Data are paired and come from the same population.
2. Each pair is chosen randomly and independently.

3. The data are measured on at least an interval scale when, as is usual, within-pair differences are calculated to perform the test.

In Wilcoxon test, equation for test statistic W is as follows

$$W = \sum_{i=1}^{N_r} [sgn(x_{2,i} - x_{1,i}).R_i] \tag{5.6}$$

Where x_2 and x_1 are observations computed using Set_1 and Set_2 metrics respectively. R_i denotes the rank. The two-sided test consists in rejecting H_0 if $|W| > W_{critical,N_r}$. In our case we set $\alpha = 0.05$.

5.5 Experimental Methodology

Keeping in view our objective and skewed datasets, we adopted an experimental methodology which is used in various studies of SFP[6, 23, 29, 32, 34, 36, 41, 46, 49, 51, 53, 55, 57, 59, 209].

We first perform the ULR to compute the significance of the included coupling metrics individually. The significant metrics are later assessed for the existence of multicollinearity.

Multicollinearity is a statistical concept where two independent variables in a model are correlated. Multicollinearity can lead to skewed or misleading results. Two variables are considered to be perfectly collinear if their correlation coefficient is +/- 1.0. This process would be done using Spearman correlation coefficient. In statistics, Spearman's rank correlation coefficient or Spearman's ρ, is a nonparametric measure of rank correlation. It assesses how well the relationship between two variables can be described using a monotonic function. It is better to use independent variables that are not correlated or drop the strongly correlated variables. Later the least correlated metrics are used to build a MLR model.

In MLR is the outcome variable (dependent variables) is dichotomous (e.g., *fp* or *nfp*). Its aim is to derive the best-fitting model to describe the relationship between an outcome and a set of predictors. Here, the independent variables are called coupling metrics and dependent variable is a fault label, i.e. *fp* and *nfp*.

The performance of MLR would be assessed using F1-measure. F-measure is the harmonic mean between Precision and TPR, first introduced in [164]. It tells how precise a classifier is (how many instances it classifies correctly), as well as how robust it is. Mathematically it is written as.

$$F\text{-}measure = \frac{(\beta^2 + 1) \times Precision \times TPR}{\beta^2 \times Precision + TPR} \tag{5.7}$$

Where, β can be 0.5, 1, and 2 for F0.5, F1, and F2 measures respectively. Varying the value of β allows different weights to be assigned to false negative and false positive. In our work, a value of $\beta = 1$ has been used, which equalizes the influence of false negatives and false positives. Since declaring *fp* module as *nfp* has a performance overhead. Whereas, declaring a *nfp* module as *fp* incurs greater testing cost. We need to have a balance between false negative and false positive. Therefore, F1 score is more meaningful than the F0.5 and F2 score in the current context. Figure 5.2 graphically illustrates the methodology. The reason is that

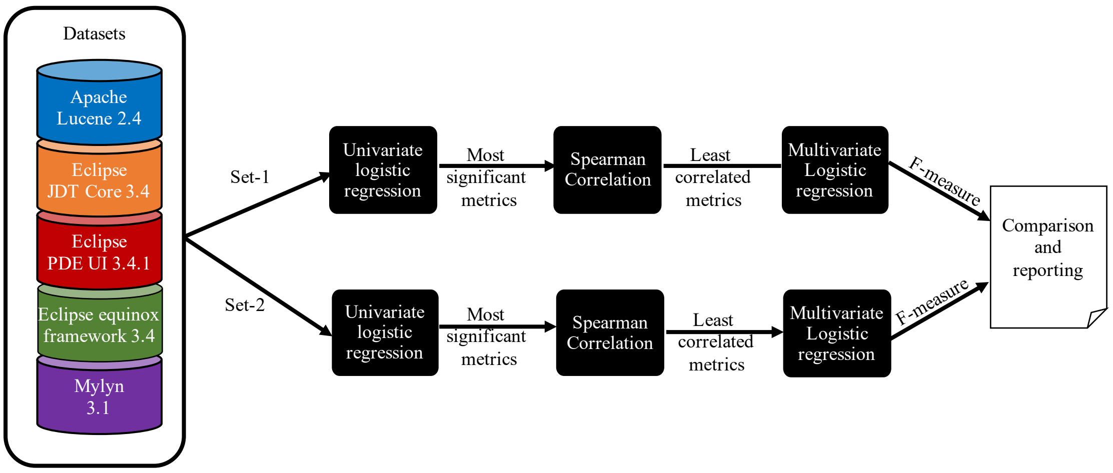


FIGURE 5.2: Experimental methodology

these algorithms are least susceptible to imbalance datasets[239]. We performed 10 experiments using the set of independent variables (see Table 5.2).
In all the cases, dependent variable is binary, which shows the *fp* or *nfp* classes. To avoid overfitting in ULR and MLR, we drew a random sampling, early stopping and finally perform 10-fold cross-validation for training and validation purposes. Finally, the average model is used (to avoid biased) on testing data, wherein F1-Score are computed.

This is the most common dependent variable used in 70% of the SFP studies [22].

We applied MLR to build model. Each time, we split the datasets for training and testing purposes. After that, we performed 10-folds cross-validation on the training set.

Table 5.2: Descriptions of the experiments performed

Sr. No.	**Datasets**	**IV**	**DV**	**Algorithm**	**Performance measure**
1	Apache	Set_1	Binary (fp nd nfp)	MLR	F-measure
2	Lucene 2.4	Set_2			
3	Eclipse equinox framework 3.4	Set_1			
4		Set_2			
5	Eclipse JDT Core 3.4	Set_1			
6		Set_2			
7	Eclipse PDE UI 3.4.1	Set_1			
8		Set_2			
9	Mylyn 3.1	Set_1			
10		Set_2			

Chapter 6

Experimentation and Results

We performed the methodology which has been discussed in Chapter 5. In this chapter, we report the results computed in each phase. Finally, there is a discussion on the results and briefly discussed threats to validity.

6.1 Results of Univariate Logistic Regression

A ULR is undertaken using seven coupling metrics (i.e. Ce, CBO, Fan-in, Fan-out, RFC, Vovel-in, and Vovel-out) one after another against the dependent variable, i.e., *fp* and *nfp*. Table 6.1 shows the coefficient computed and the p-value for all seven coupling metrics at $\alpha = 0.05$.

TABLE 6.1: Results of the ULR using coupling metrics in the selected five datasets

Metrics	Params.	Apache Lucene	Eclipse equinox	Eclipse JDT Core	Eclipse PDE UI	Mylyn 3.1
Ce	Coeff.	0.051	0.01	0.015	0.01	0.12
	p-value	0	0	0	0	0
CBO	Coeff.	0.066	0.02	0.042	0.1	0.101
	p-value	0	0	0	0	0
Fan-in	Coeff.	0.028	0.014	0.017	0.046	0.087
	p-value	0	0.001	0	0	0
Fan-out	Coeff.	0.357	0.24	0.274	0.278	-0.06
	p-value	0	0	0	0	0.268
RFC	Coeff.	0.16	0.13	0.16	0.18	0.03
	p-value	0	0.039	0	0	0.01
Vovel-in	Coeff.	0	0	0.001	0.001	0.001
	p-value	0	0.003	0	0	0
Vovel-out	Coeff.	0	0	0	0	0
	p-value	0	0	0	0	0.043

6.2 Results of Correlation Analysis

The correlation analysis aims to determine empirically whether the included metrics are in consonance with the other metrics or not. The strong association implies the coverage of duplicate information. Figure 6.4 through 6.17 shows the correlation result in the five datasets. The value of r for each metric pair is shown at the top of each scatter plot with their names.

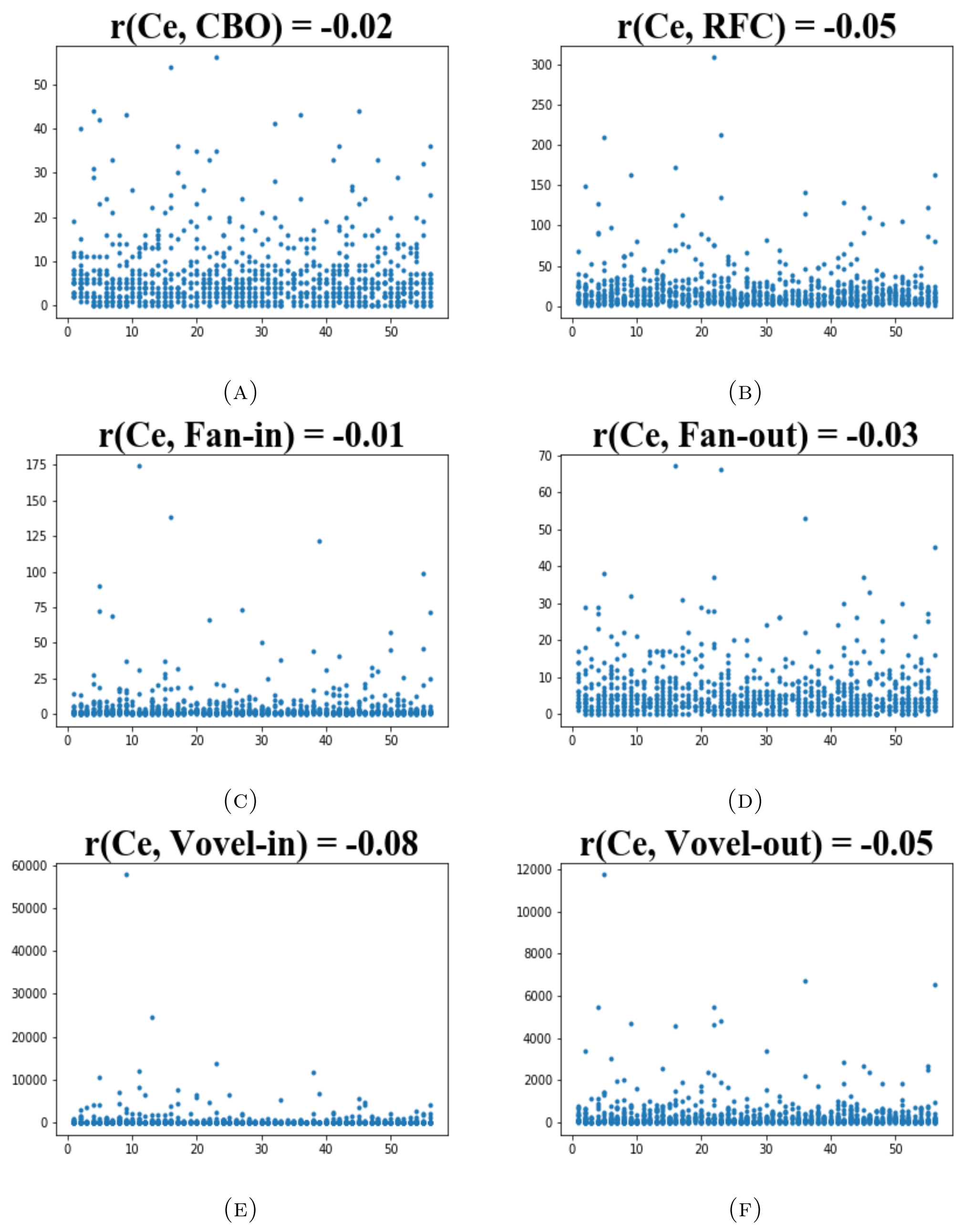
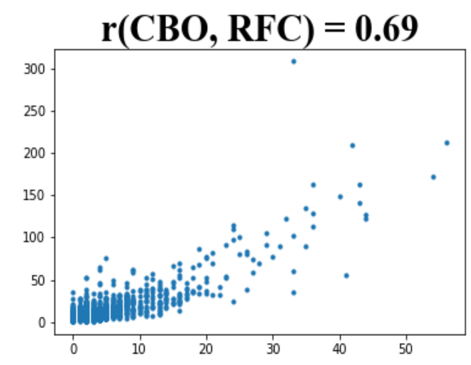
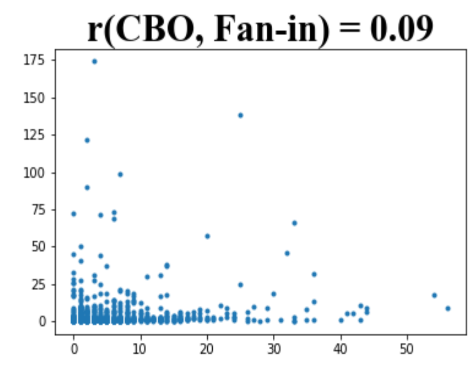


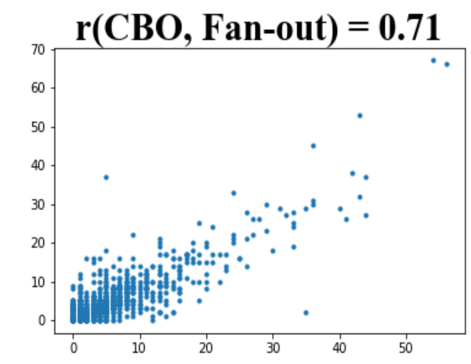
FIGURE 6.1: Correlation between pair metrics in Apache Lucene 2.4 dataset (Continue)



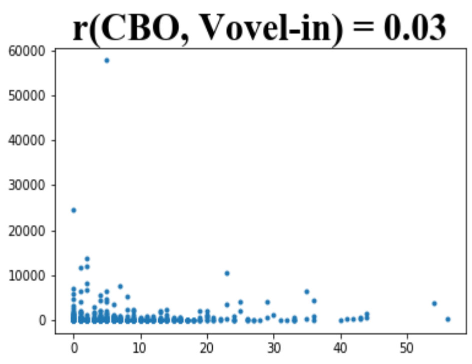
(A)



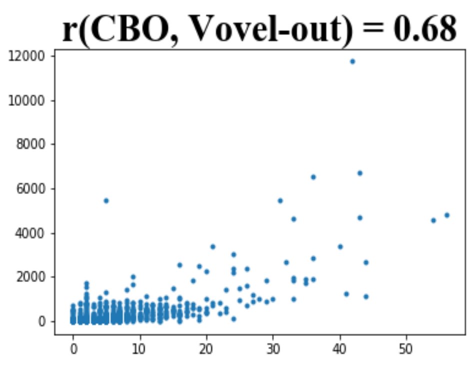
(B)



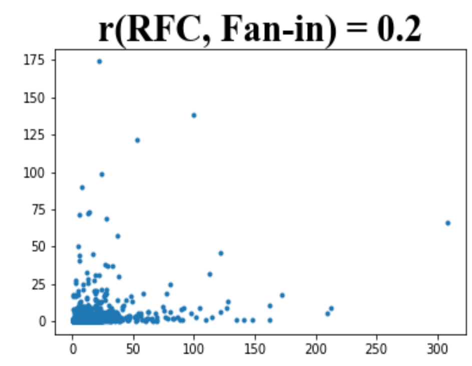
(C)



(D)

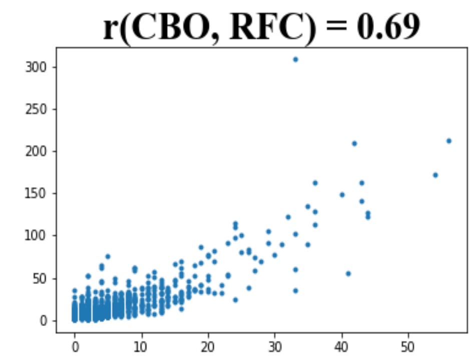


(E)

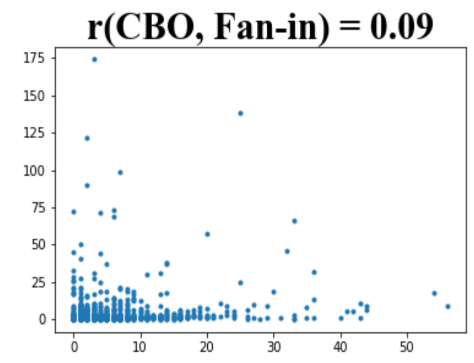


(F)

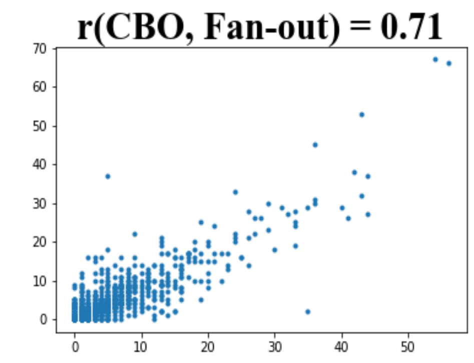
FIGURE 6.2: Correlation between pair metrics in Apache Lucene 2.4 dataset (Continue)



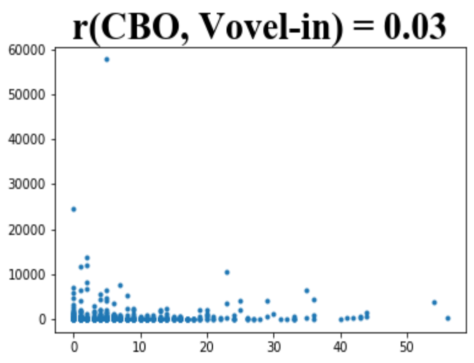
(A)



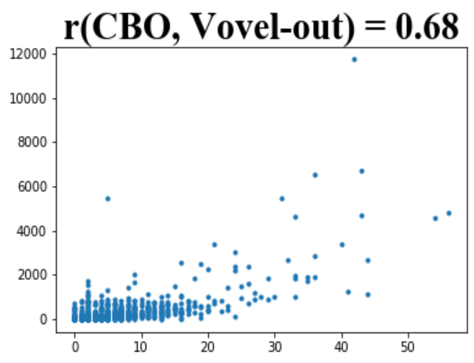
(B)



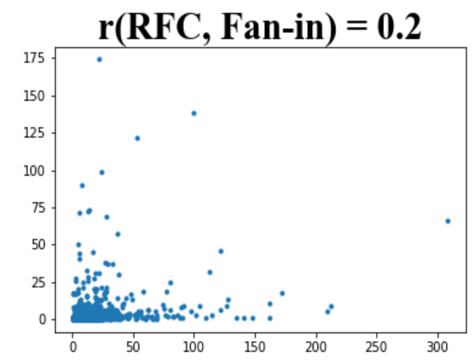
(C)



(D)

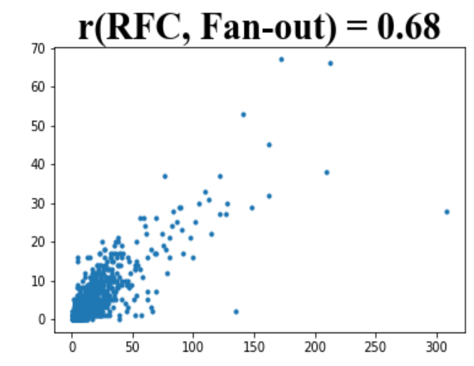


(E)

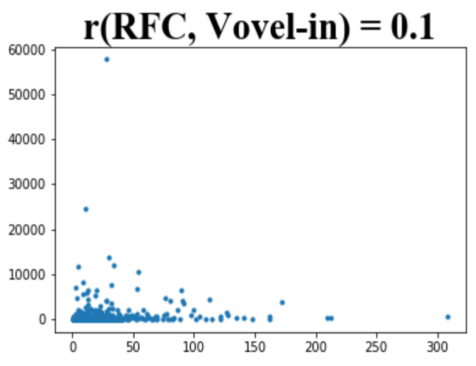


(F)

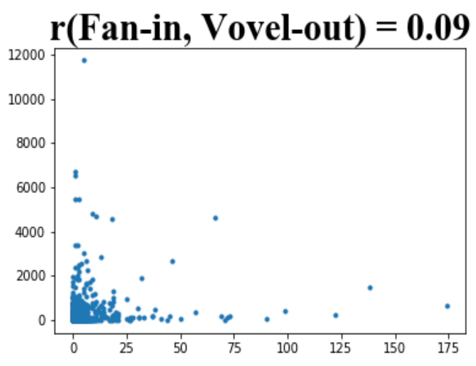
Figure 6.3: Correlation between pair metrics in Apache Lucene 2.4 dataset (Continue)



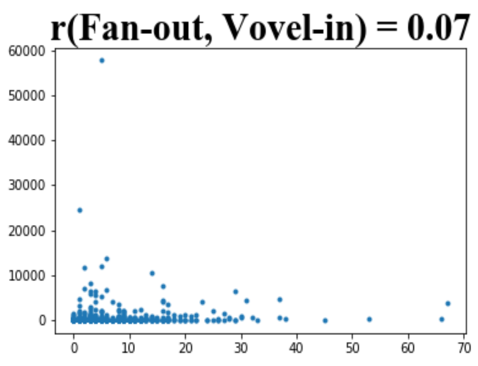
(A)



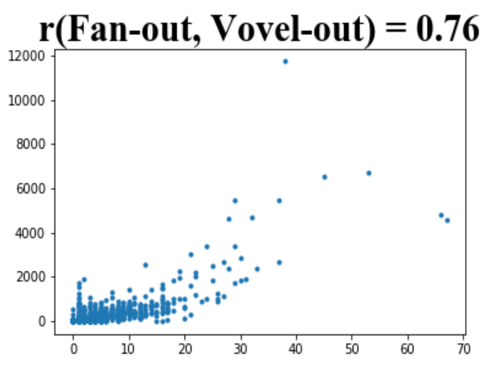
(B)



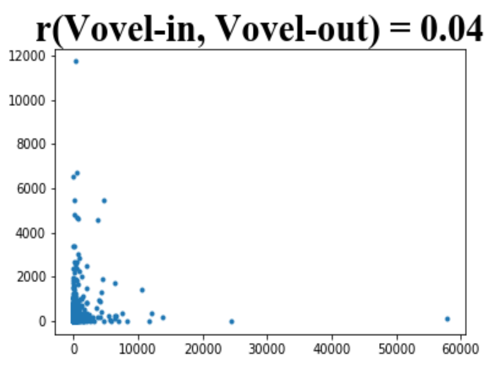
(C)



(D)

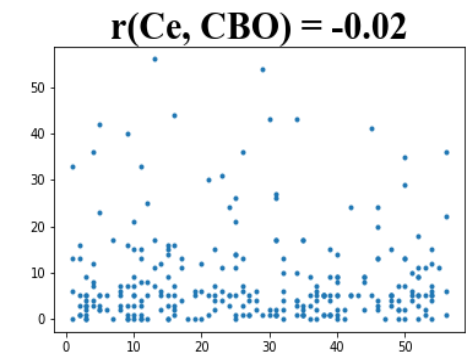


(E)

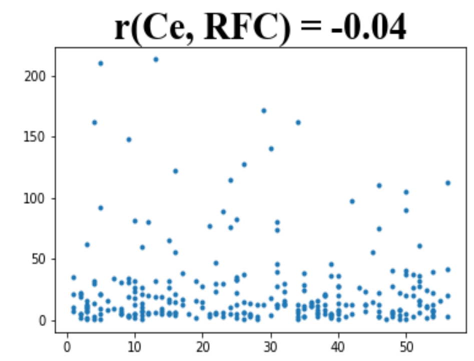


(F)

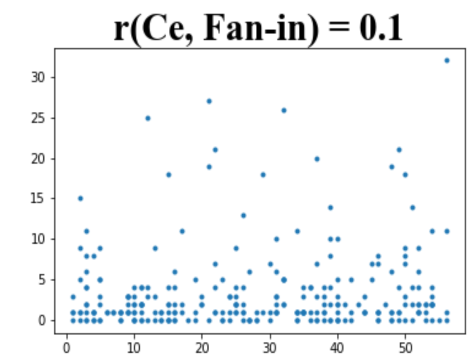
FIGURE 6.4: Correlation between pair metrics in Apache Lucene 2.4 dataset (Continue)



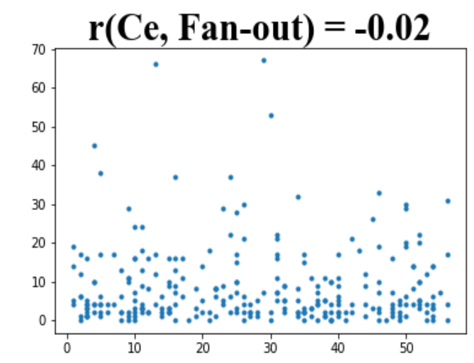
(A)



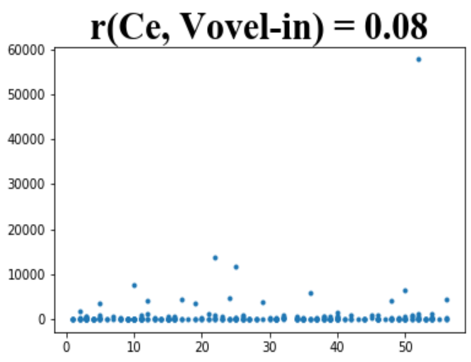
(B)



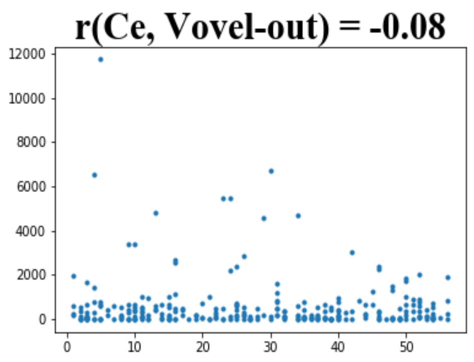
(C)



(D)

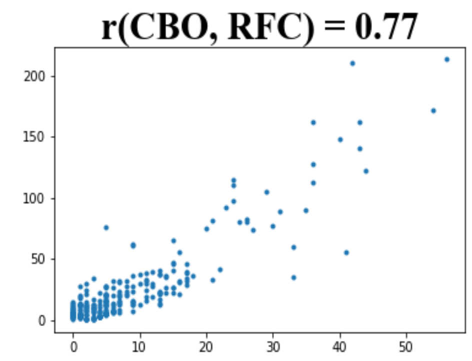


(E)

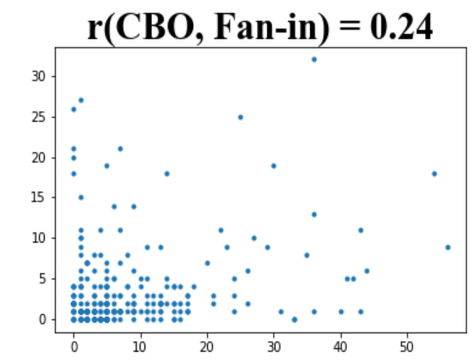


(F)

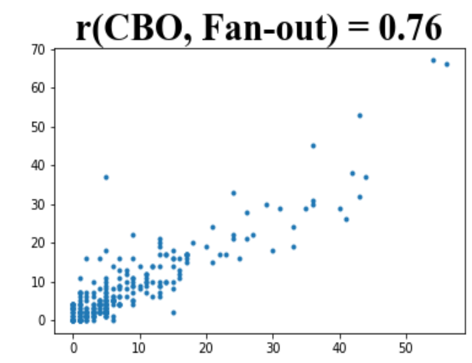
Figure 6.5: Correlation between pair metrics in Eclipse Equinox Framework 3.4 dataset (Continue)



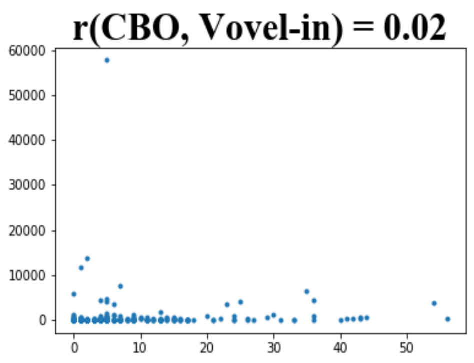
(A)



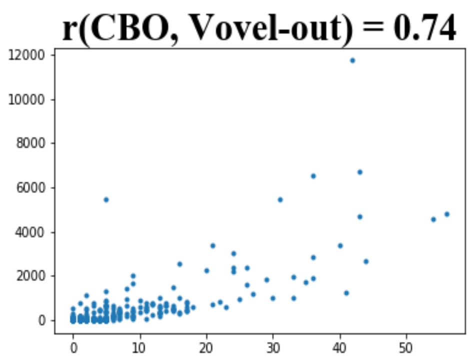
(B)



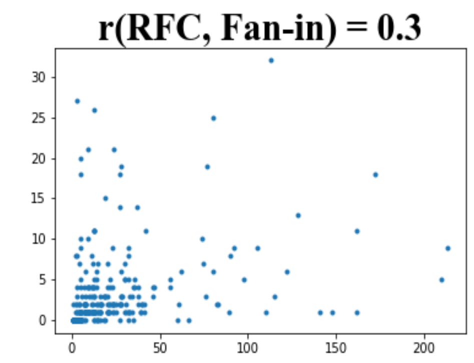
(C)



(D)

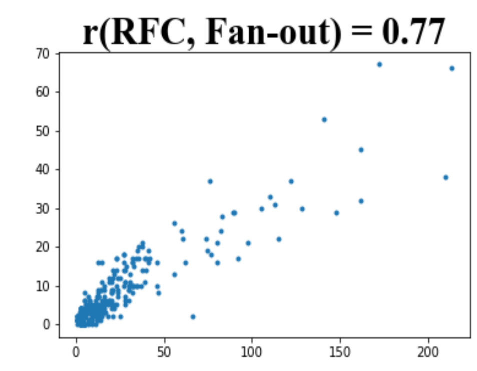


(E)

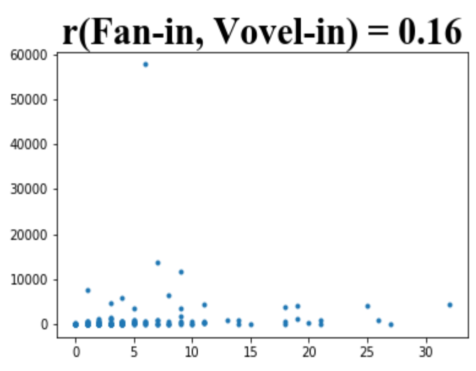


(F)

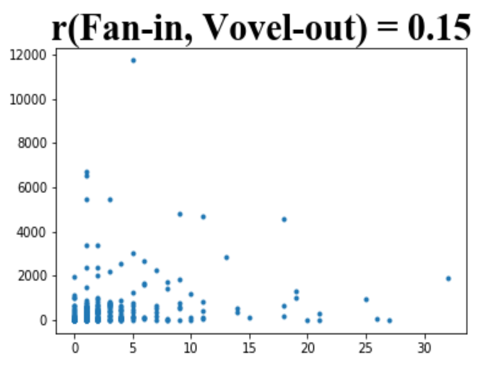
FIGURE 6.6: Correlation between pair metrics in Eclipse Equinox Framework 3.4 dataset (Continue)



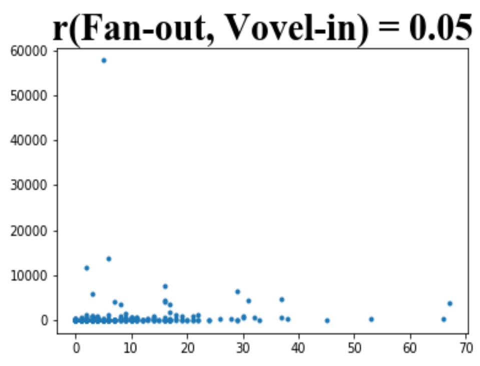
(A)



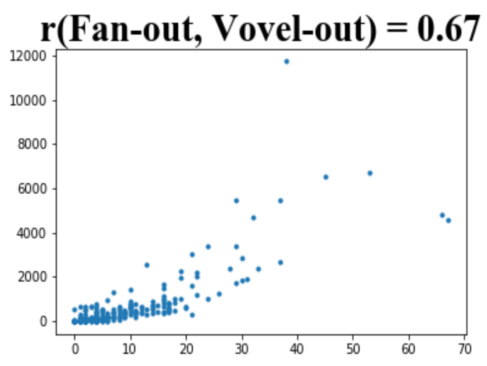
(B)



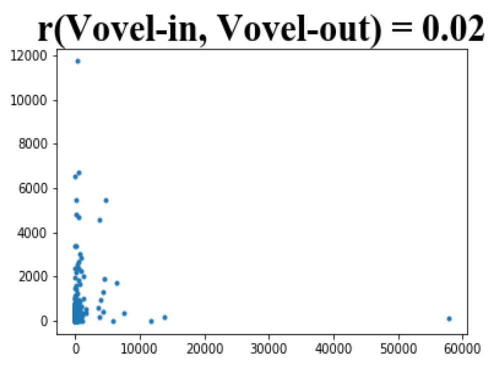
(C)



(D)

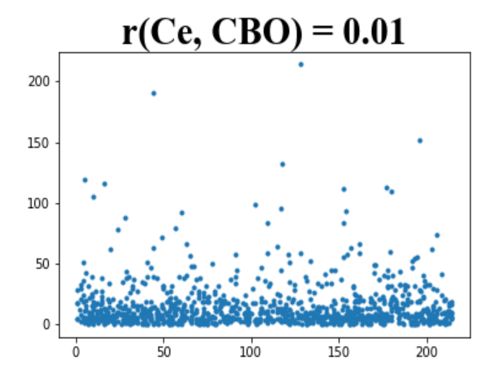


(E)

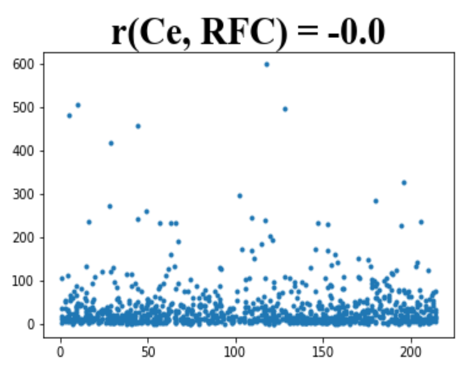


(F)

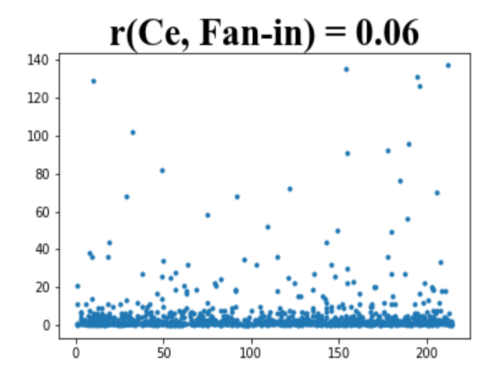
FIGURE 6.7: Correlation between pair metrics in Eclipse Equinox Framework 3.4 dataset (Continue)



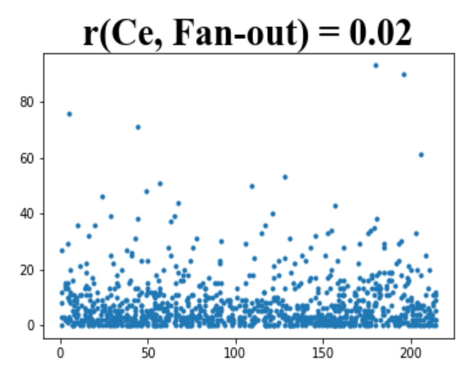
(A)



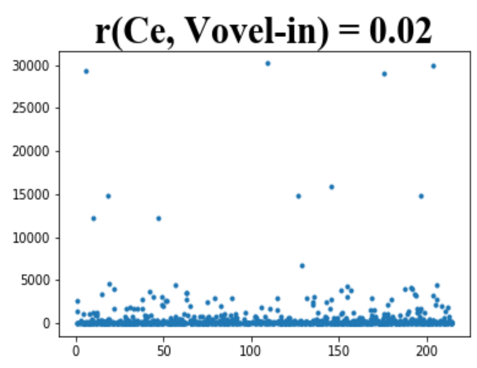
(B)



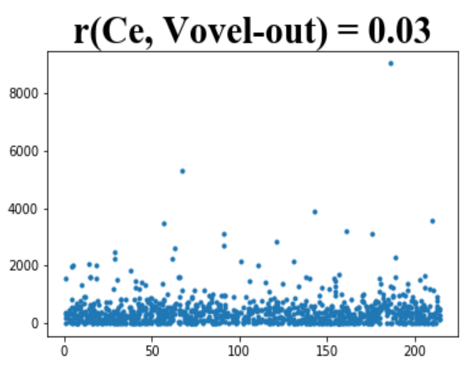
(C)



(D)

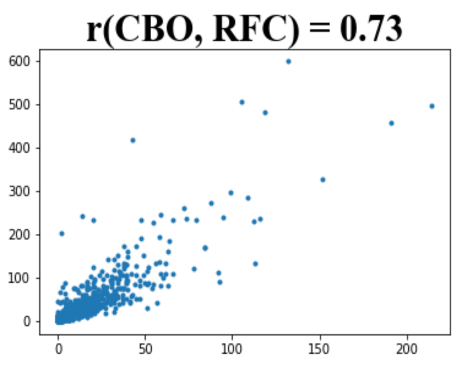


(E)

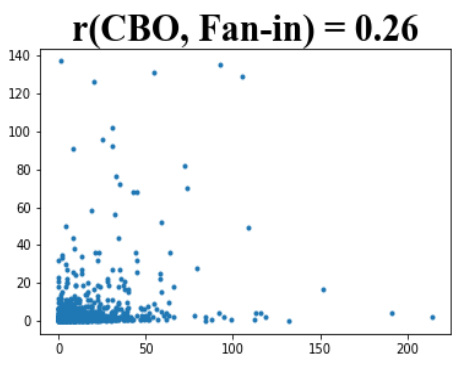


(F)

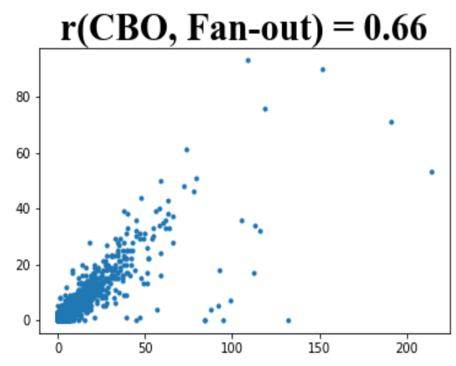
FIGURE 6.8: Correlation between pair metrics in Eclipse JDT Core 3.4 dataset (Continued)



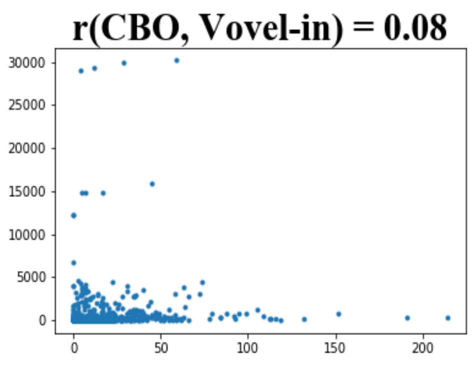
(A)



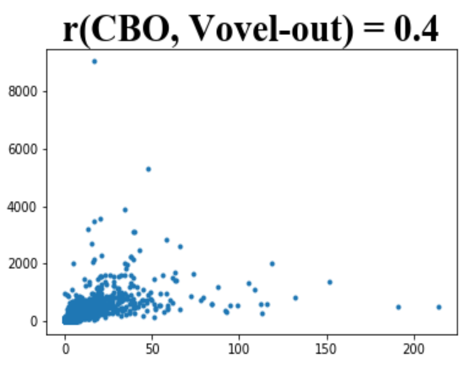
(B)



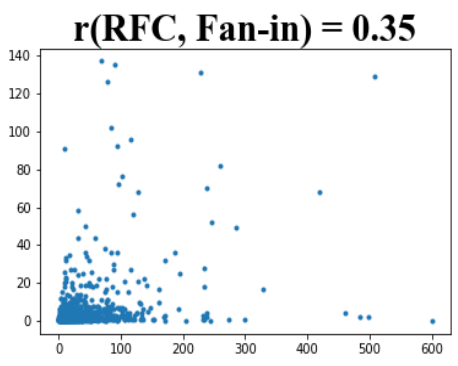
(C)



(D)

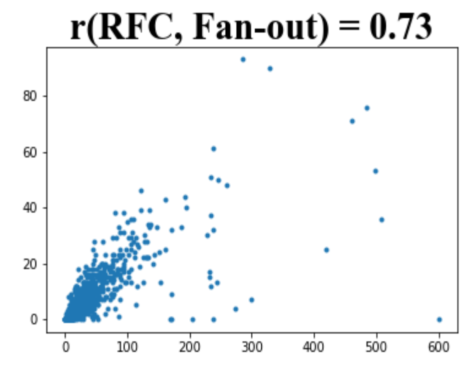


(E)

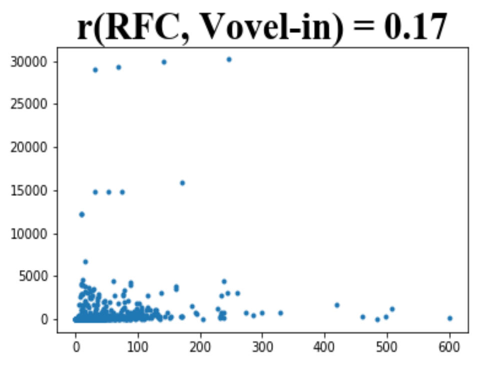


(F)

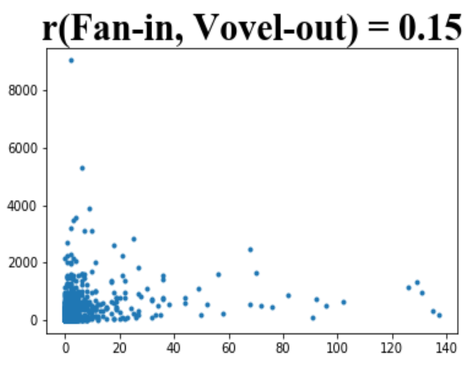
FIGURE 6.9: Correlation between pair metrics in Eclipse JDT Core 3.4 dataset (Continued)



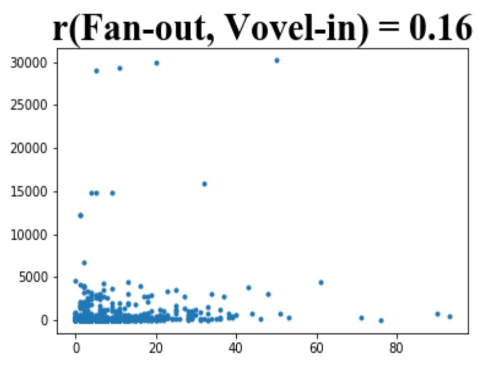
(A)



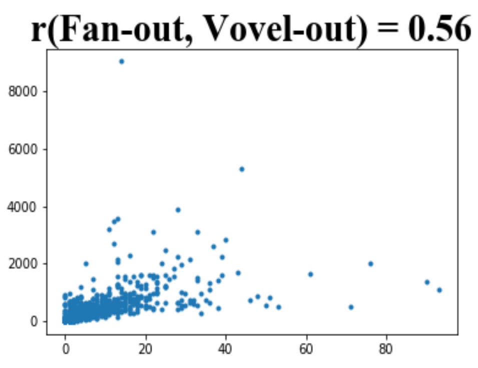
(B)



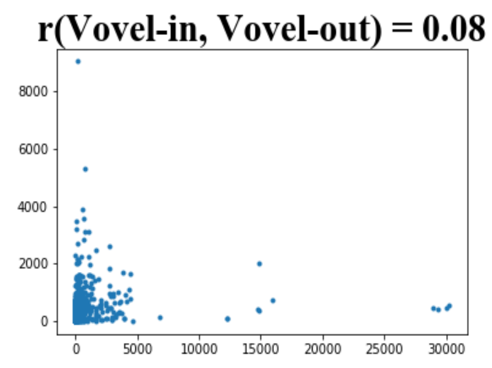
(C)



(D)

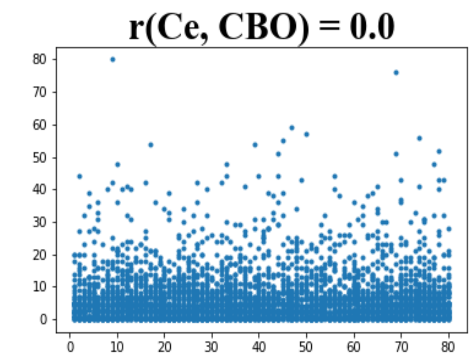


(E)

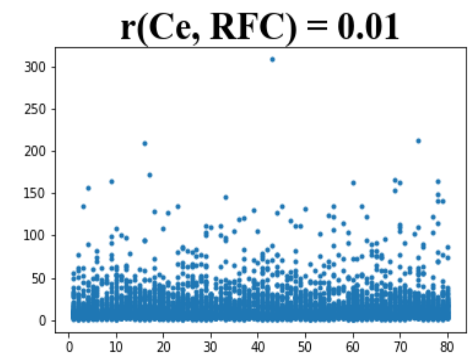


(F)

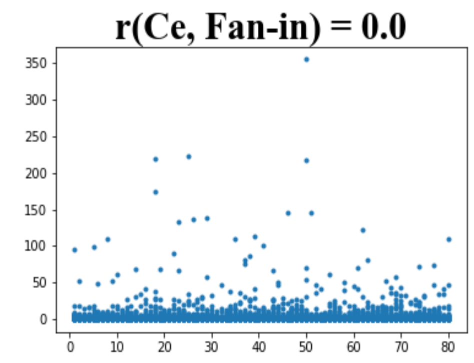
FIGURE 6.10: Correlation between pair metrics in Eclipse JDT Core 3.4 dataset (Continued)



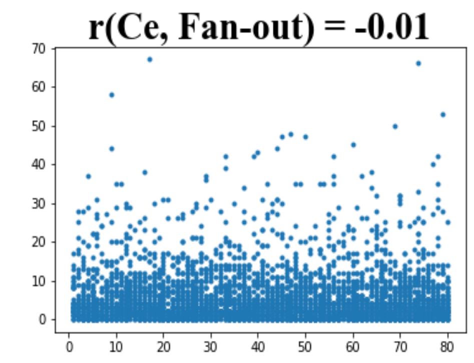
(A)



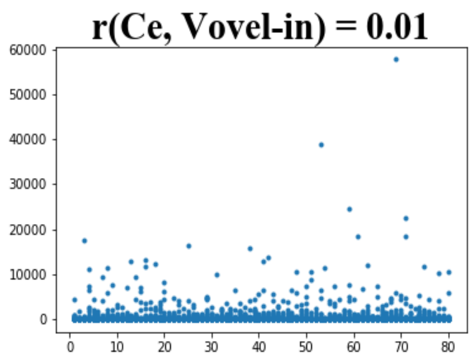
(B)



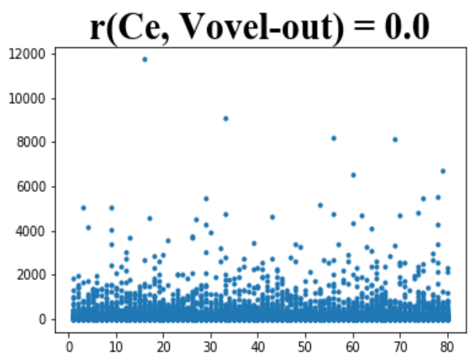
(C)



(D)

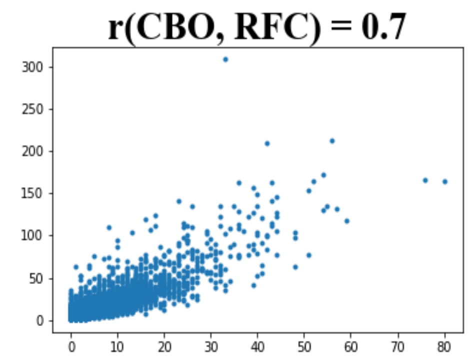


(E)

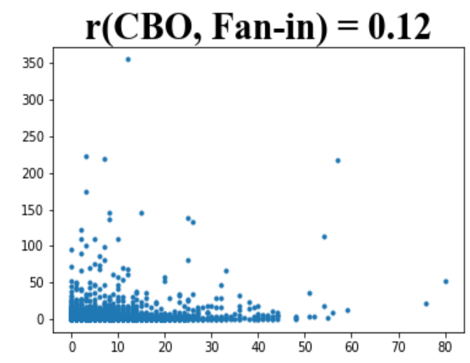


(F)

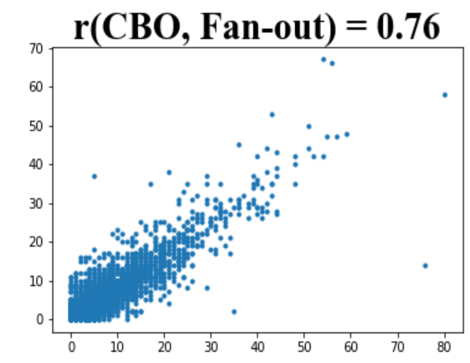
FIGURE 6.11: Correlation between pair metrics in Eclipse PDE UI 3.4.1 dataset (Continued)



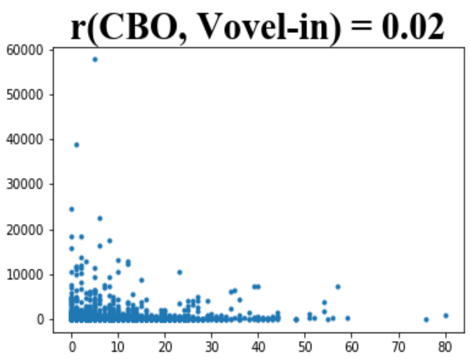
(A)



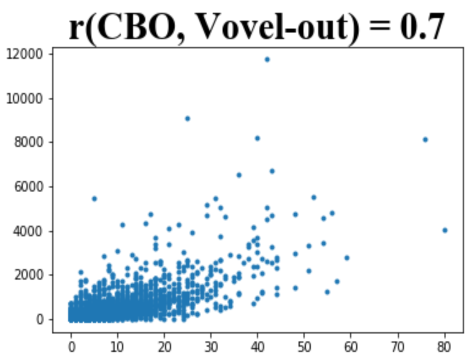
(B)



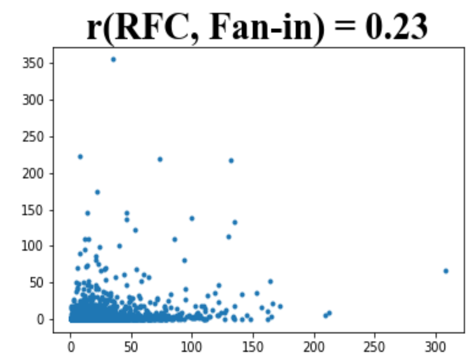
(C)



(D)

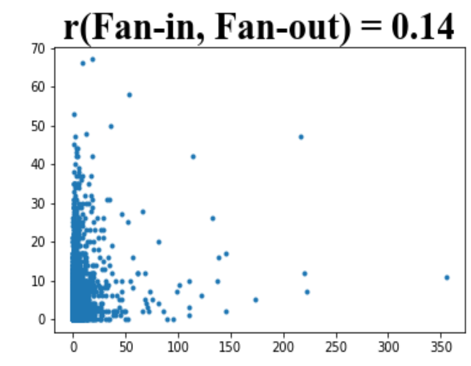


(E)

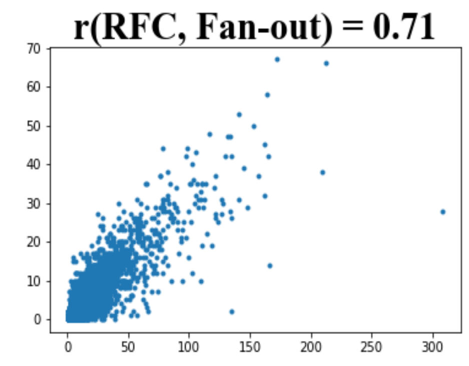


(F)

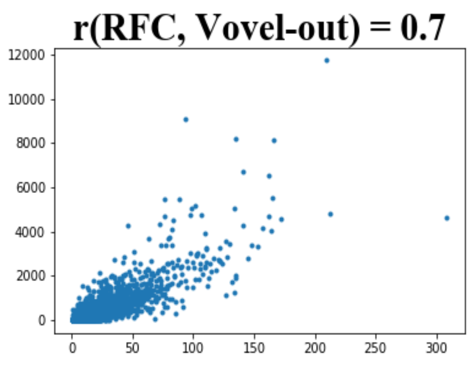
FIGURE 6.12: Correlation between pair metrics in Eclipse PDE UI 3.4.1 dataset (Continued)



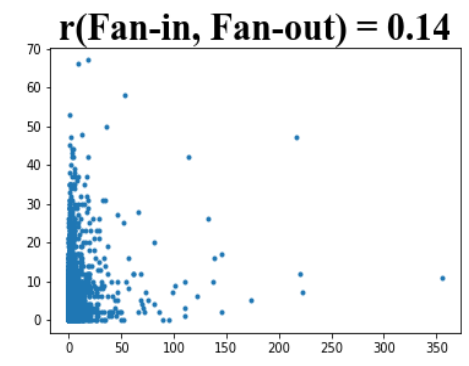
(A)



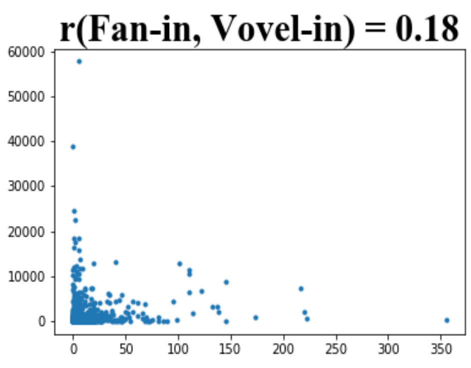
(B)



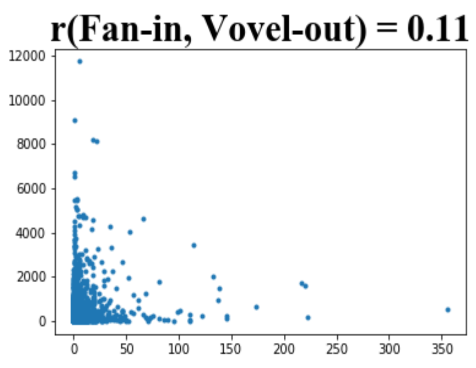
(C)



(D)

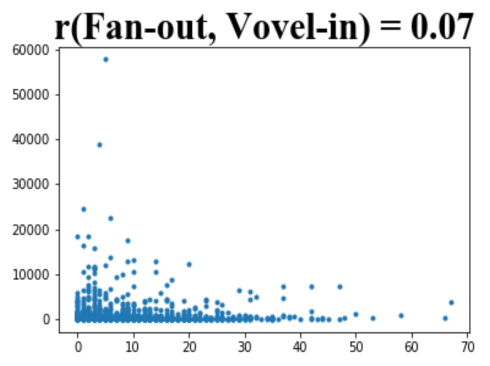


(E)

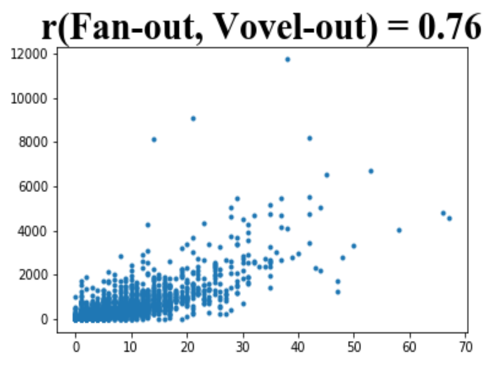


(F)

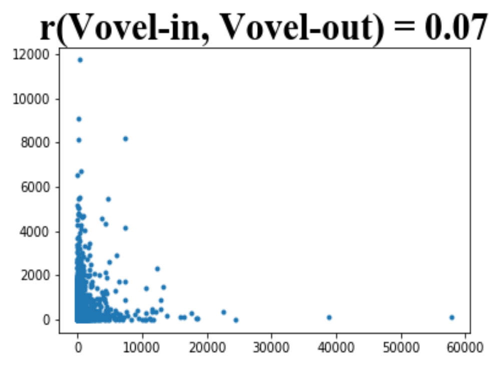
FIGURE 6.13: Correlation between pair metrics in Eclipse PDE UI 3.4.1 dataset (Continued)



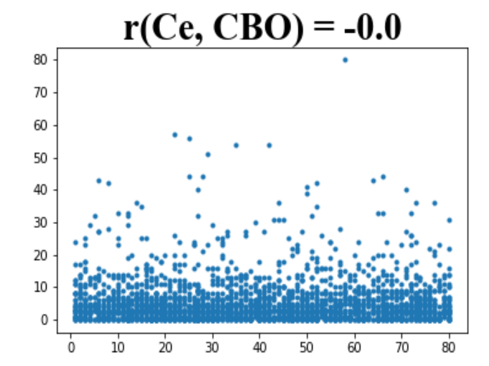
(A)



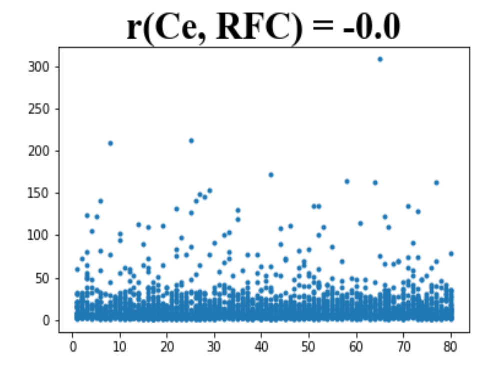
(B)



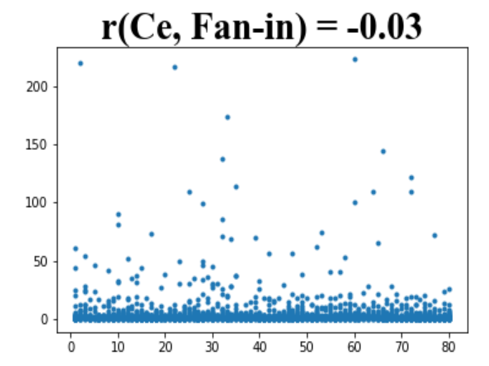
(C)



(D)

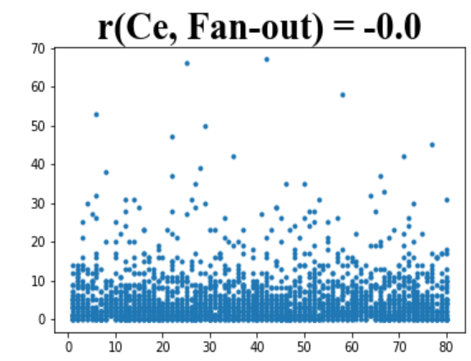


(E)

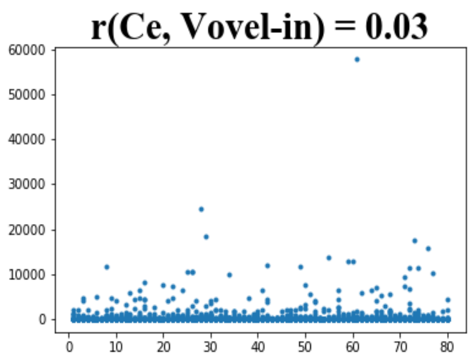


(F)

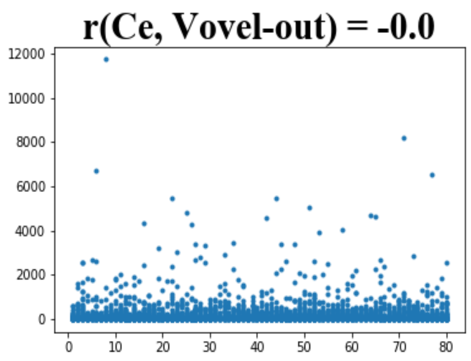
FIGURE 6.14: Correlation between pair metrics in Mylyn 3.1 dataset



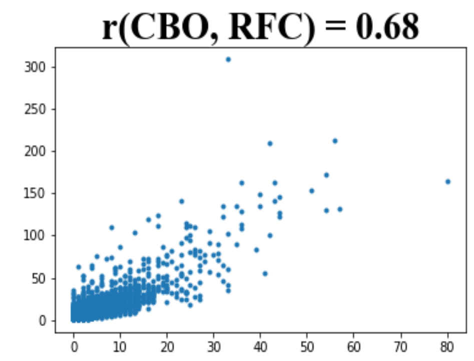
(A)



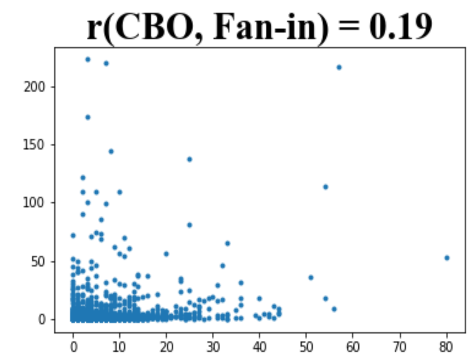
(B)



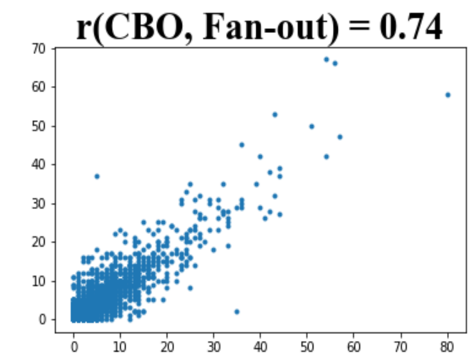
(C)



(D)

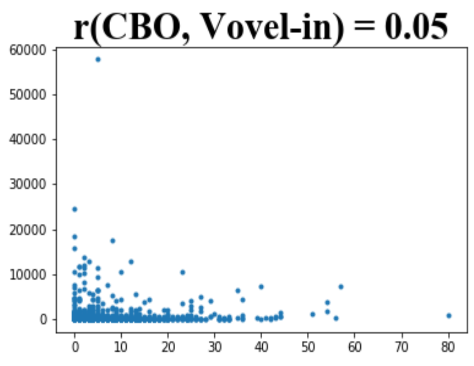


(E)

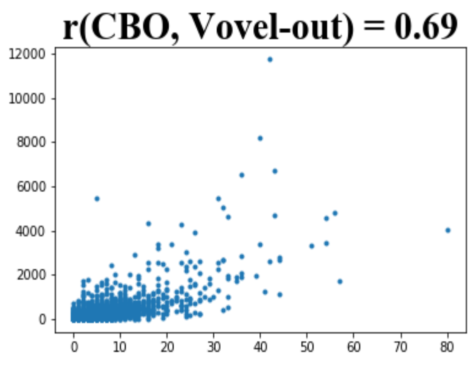


(F)

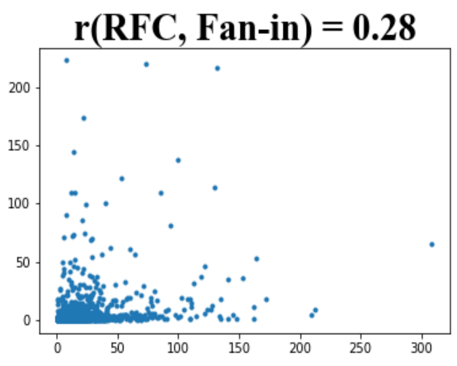
FIGURE 6.15: Correlation between pair metrics in Mylyn 3.1 dataset (Continued)



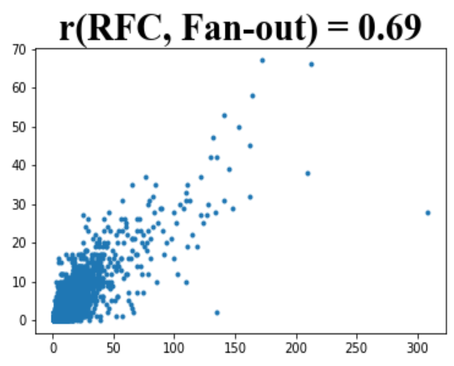
(A)



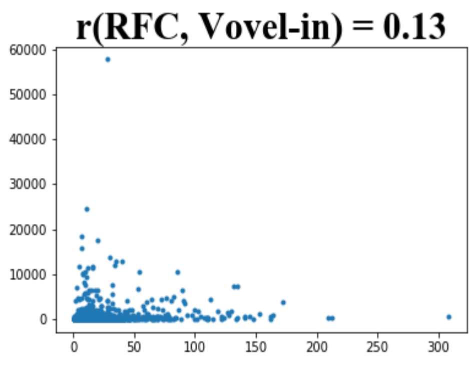
(B)



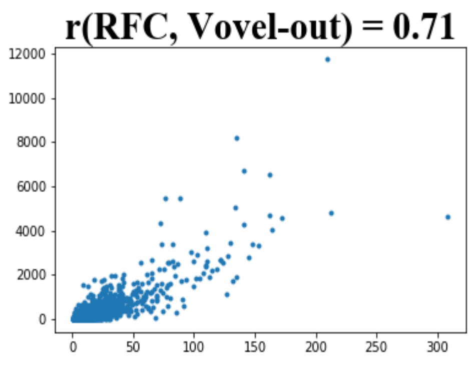
(C)



(D)

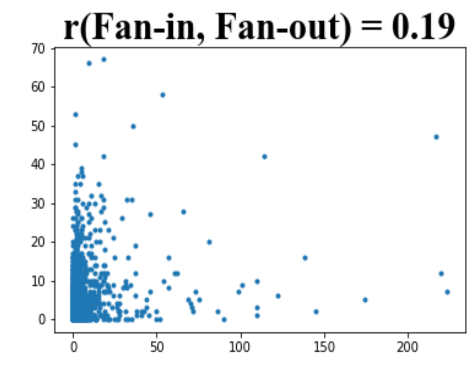


(E)

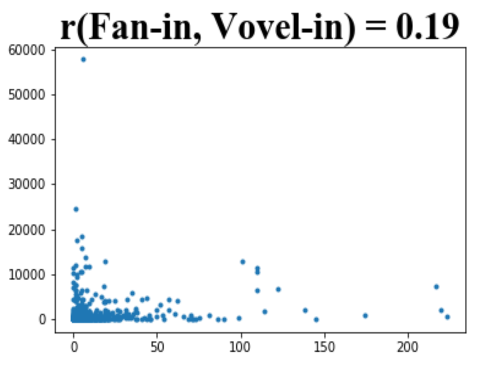


(F)

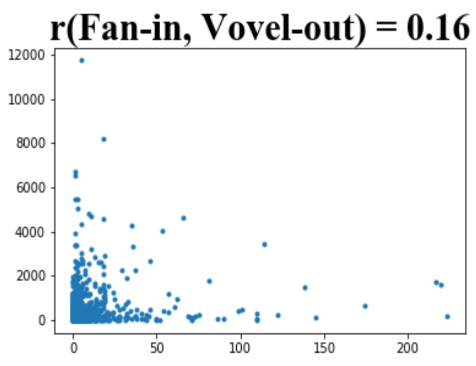
FIGURE 6.16: Correlation between pair metrics in Mylyn 3.1 dataset (Continued)



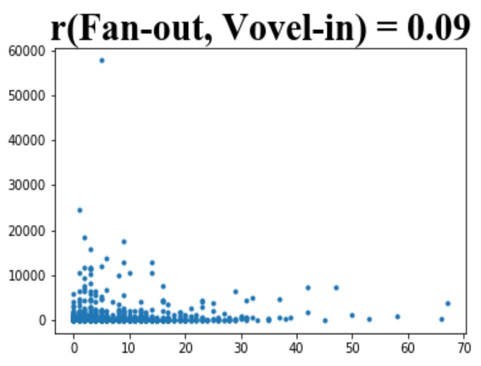
(a)



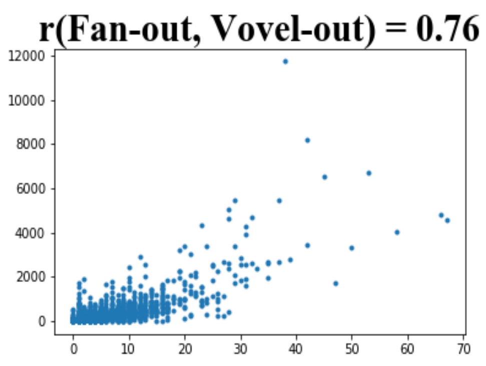
(b)



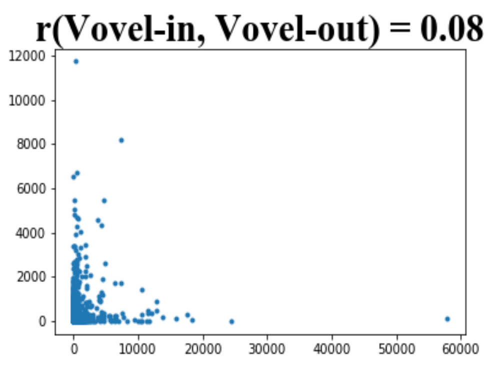
(c)



(d)



(e)



(f)

Figure 6.17: Correlation between pair metrics in Mylyn 3.1 dataset (Continued)

The scatter plots shown in Figure 6.4 through Figure 6.17 show that there exists a weak correlation between pairs of seven selected metrics. However, a mild positive correlation can be observed in all the datasets between two pairs, i.e., (RFC, Vovel-out) and (Fan-out, Vovel-out). The obvious reason is the commonality in directional coverage between these pairs. Likewise, there exists relatively weak outer data flow between modules in the selected dataset. Besides these pairs, significantly weak correlation have been observed in the rest of the pairs. This reflects the unique information coverage by the seven included coupling metrics.

6.3 Results of Multivariate Logistic Regression

While considering the experimental methodology described in the last chapter, MLR is applied in 10 different experiments by varying independent variables and datasets. Finally, the average model is run on the test set for computing F-measure. The results are shown in Table 6.2. The last two columns show the results of Wilcoxon test with their corresponding coefficient and p-values.

Table 6.2: MLR results in five datasets using Set_1 and Set_2 metrics

Datasets	F-measure in Set_1	F-measure in Set_2	Coeff.	p-values
Apache Lucene 2.4	0.71	0.74	0.71	0.0000
Eclipse Equinox Framework 3.4	0.63	0.76	0.63	0.0002
Eclipse JDT Core 3.4	0.46	0.67	0.46	0.0004
Eclipse PDE UI 3.4.1	0.57	0.60	0.57	0.0004
Mylyn 3.1	0.76	0.81	0.76	0.0004

The above results are computed at the best threshold of probabilistic function. However, the results of True positive rate and False positive rate at different thresholds from 0.0 to 1.0 have also been computed and shown in Figure 6.18 through Figure 6.22.

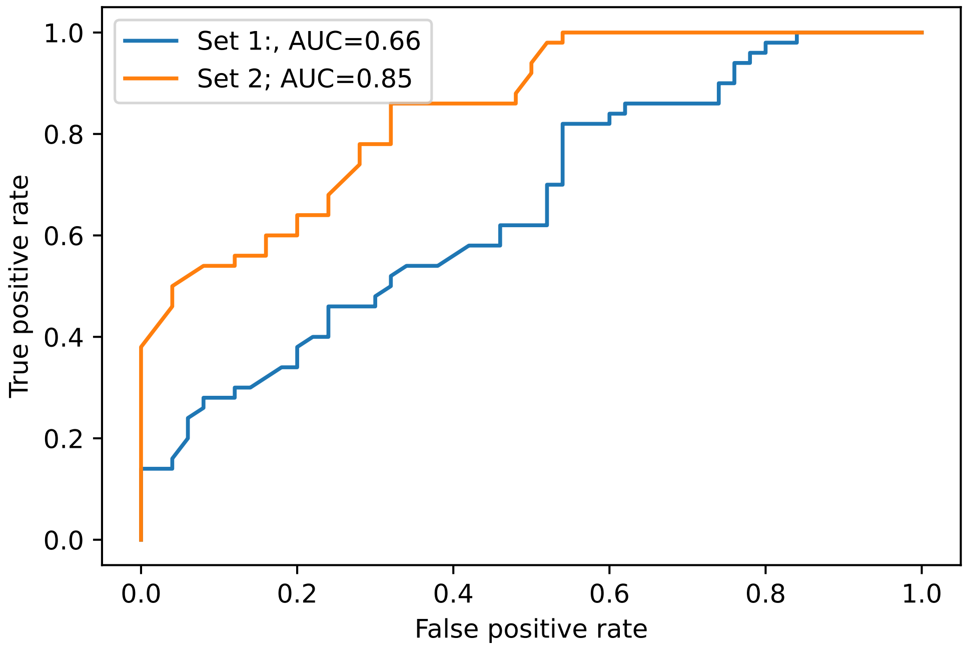


FIGURE 6.18: AUC in Apache Lucene 2.4

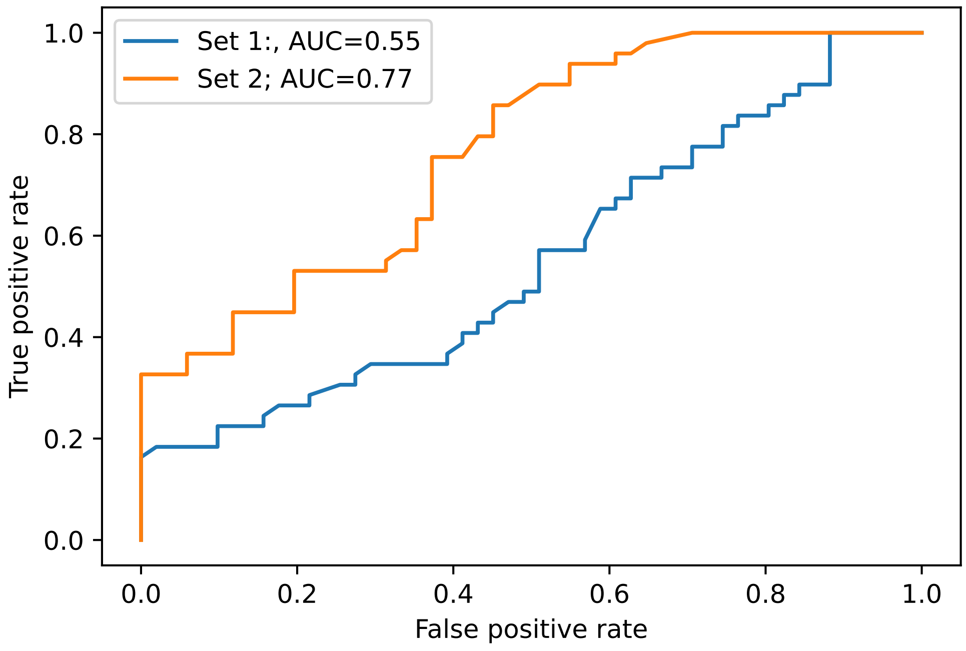


FIGURE 6.19: AUC in Eclipse Equinox Framework 3.4

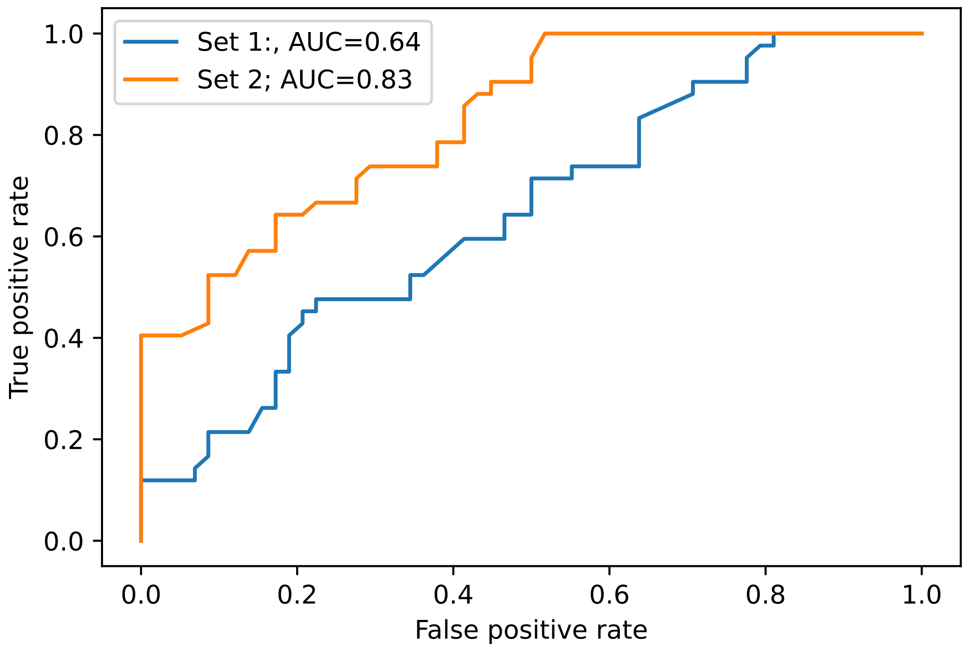


FIGURE 6.20: AUC in Eclipse JDT Core 3.4

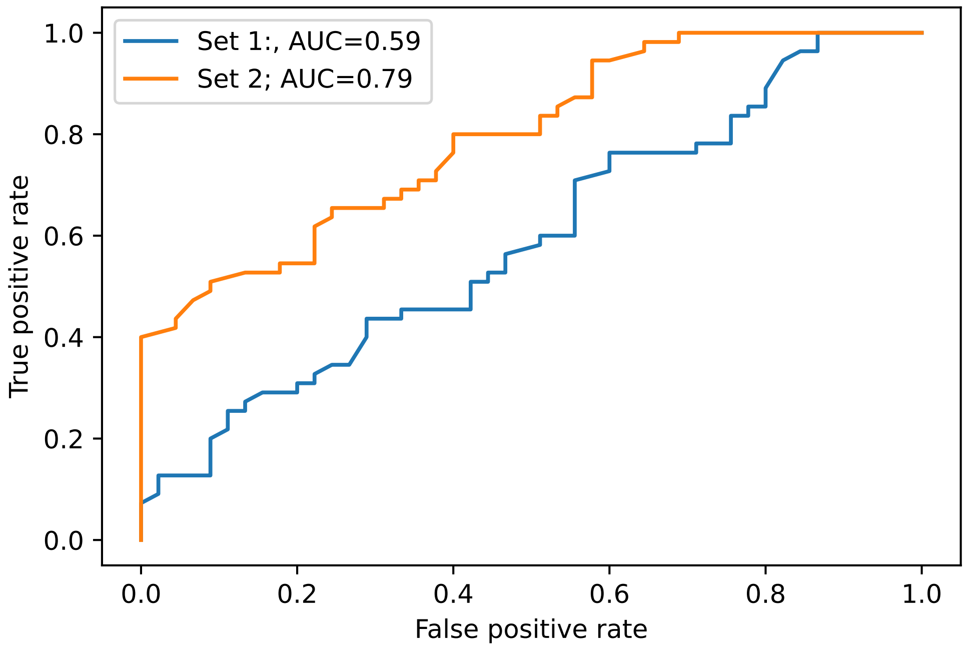


FIGURE 6.21: AUC in Eclipse PDE UI 3.4.1

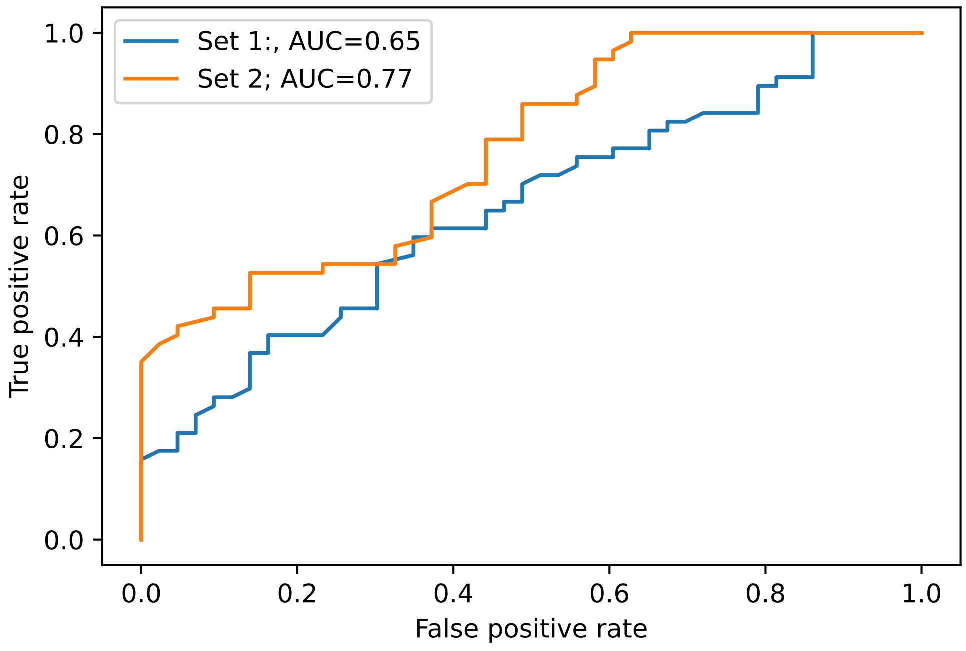


FIGURE 6.22: AUC in Mylyn 3.1

The computed AUC shows the outperformance of MLP classifier using Vovel metrics in all the five datasets.

6.4 Discussion on Results

The results of ULR advocate that the included seven coupling metrics have a significant association with fault- proneness. The results are similar to the conclusion drawn by other studies [36, 52, 53, 207]. More specifically Vovel metrics have lower coefficients than that of the other metrics. This implies the stronger association of the proposed Vovel metrics in SFP than its counterparts. Likewise, it shows the importance of data flow volume and coupling level in SFP. The results of the Spearman correlation coefficient illustrate the following conclusion.

1. The Vovel-in and Vovel-out metrics are having a very weak relationship between them in all the five datasets. This represents the orthogonal nature of the proposed metric w.r.t. each other.

2. Vovel-in is least correlated with CBO in all the five datasets.

3. Mild association is observed between Vovel-out and Fan-out likewise Vovel-out and RFC. The obvious reason is the consideration of direction of method calls by the corresponding associating metrics.

4. There exists a weak correlation (i.e., < 0.8) between pair of metrics in all the datasets. This implies significant exclusive information coverage by the included metrics.

Eventually, a weak association between pair metrics across the datasets represents the unique information coverage by all the metrics. Therefore, none of the metrics has been dropped, and all the metrics are included in the final phase i.e. MLR. The experiment using MLR directly relates to our formulated null hypothesis. The experiment illustrates the effectiveness of proposed metrics in SFP. The results infer the outperformance of Vovel metrics in all five datasets.
More specifically, in Eclipse JDT Core datasets, the performance has been improved to 0.21 on the F-measure score. Whereas a mild performance improvement (i.e, 0.03) is observed in Apache Lucene, and Eclipse PDE UI datasets.
Wilcoxon test is used for these assessments, which certifies the significance of the results. Hence, the null hypothesis that *the data flow volume and coupling levels do not improve the performance of SFP when used in combination with existing coupling metrics* should be rejected, while accepting the alternate hypothesis. Now we can safely state that the proposed Vovel metrics significantly improve the performance of SFP when used in combination with existing coupling metrics. Moreover, keeping in view the objective of our study, we can conclude that data flow volume and coupling levels improve the performance of SFP.

6.5 Threats to Validity

The results of our experiment allow us to associate Vovel metrics with SFP. Nevertheless, before we could accept the result, we would have to consider possible threats to its validity.

1. We include the content coupling in our proposed metrics, however, we could not parse it due to its difficult nature. The reason is that it involves the manipulation of pointers or residences of two modules physically in one compilable entity. Moreover, it can also occur by just sharing the data area of multiple modules. If we would do so, the results will even be more promising. Hence, the impact of content coupling in SFP remains unrevealed.

2. Concerning the size of the projects, sufficient comprehensible project size is taken. The projects of a very large size or very small size were ignored. The reason was the unavailability of either projects' source code or fault information.

3. The selected open-source projects are developed in Java, which sufficiently justifies the objective of the experiment and successfully demonstrates the experimental methodology. However, since the coupling aspects vary in different programming languages. The results may vary when using projects developed in languages other than Java.

Chapter 7

Conclusion and Future work

Software coupling is an important design parameter. Software research community is quite active in computing the degree of coupling by proposing numerous coupling metrics. These metrics are utilized in various disciplines, like fault prediction, design patterns, impact analysis, re-modularization, assessing software quality, maintenance cost, productivity, software vulnerabilities, reusability, changeability, reliability, and maintainability. This thesis focuses on effective utilization of coupling metrics in the software fault prediction.
Software modules' coupling is indispensable for making the software easy to understand, develop, test, and deploy. A comprehensible level of coupling is always desirable, however, when there exists complexity in coupling in terms of levels and volume of data flow there comes incomprehensibility. Consequently, the programmer loses control and thus leads to the introduction of faults into the module under development. Hence it may be used with great care.

7.1 Conclusion

Coupling metrics are generally found useful in predicting software faults. More specifically, CBO, RFC, Ca, Fan-in, and Fan-out are reported outperformers by SFP community. However, numerous coupling metrics have not been assessed exclusively in SFP. Consequently, the exclusive factors addressed in these metrics are yet to be evaluated. Moreover, a few potentially useful factors related to coupling are also ignored. The data flow volume and coupling levels are amongst those factors.

In this research thesis, we proposed two coupling metrics, *Vovel-in*, and *Vovel-out*, that incorporate the ignored yet important aspects of coupling. *Vovel-in* captures the calls/utilization of the module while *Vovel-out* represents the calls/utilization by a module. To assess the effectiveness of the proposed metric we performed experiments. Five public datasets are taken as a case study. We utilized Java-Parser for computing Vovel metrics from the source code. First, we performed the ULR to compute the significance of the coupling metric. The significant metrics were later assessed for the existence of linear association with the Vovel metrics using Spearman correlation coefficient. The objective was to assess the presence of unique information coverage by Vovel metrics. Later the least correlated metrics were used to build a MLR model.
A ULR has been undertaken using seven coupling metrics (i.e. Ce, CBO, Fan-in, Fan-out, RFC, Vovel-in, and Vovel-out) one after another against the dependent variable, i.e., *fp* and *nfp*. The results of ULR advocated that the included seven coupling metrics have a significant association with fault- proneness. Later, in correlation assessment, it is found that there exists a weak correlation (i.e., < 0.8) between selected metrics with the Vovel metrics. This implies the significant exclusive information coverage by Vovel metrics. Finally, MLR model has been built on two sets of metrics. The first set comprises five conventional metrics, while the second metrics set comprises metrics from the first set along with the two proposed Vovel metrics. The MLP has been assessed using F1-score, while Wilcoxon test is used for significance assessments. We reject the null hypothesis that the data flow volume and coupling level do not improve the performance of SFP when used in combination with existing coupling metrics.
The outcomes of the experiment advocate the viability of the proposed metrics. The researchers and practitioners of the testing community can confidently utilize metrics Vovel metrics in identifying fault-prone modules. The proposed metrics can be helpful in drawing testing case prioritizing and can potentially assist in determining ”where to start testing from”, eventually, minimizes testing cost and time.

7.2 Future Work

The Vovel metrics are proposed both for method level coupling and class level coupling. Since we perform the evaluation of Vovel metrics at the class level and found it useful, however, the performance of the metrics at the method level needs to be evaluated also.

Besides SFP, Vovel metrics may be used for assessing software quality, maintenance cost, productivity, software vulnerabilities, reusability, changeability, maintainability, etc. the way other coupling metrics are being used.

Yourdon *et al.* [118] stated five principles of coupling, i.e., direct, local, obvious, flexible, and broad coupling. Direct coupling refers to the direct interfacing between software modules, without referring to multiple intermediate modules. Local coupling between two modules refers to the availability of all the necessary information within the connection. Obvious coupling is in reciprocal to obscure coupling wherein representation does not hide any important information. Finally, flexible coupling refers to the maintenance of one module without referring to or changing other modules. In contrast to flexible coupling, in the rigid coupling, maintenance of one module requires fixing of more than module(s) also.

In this study, Boradness of coupling due to Volume has been considered has been captured and assessed in SFP. Four other aspects of coupling stated by Yourdon *et al.* [118], i.e., direct, local, obvious, and flexible coupling are yet to be evaluated by SFP community. Therefore, in the future, the capturing and utilization of these coupling aspects can have an important aspect to do. Moreover, as it is evident from Table 3.5 most of the coupling metrics have not been addressed by the SFP community. Evaluation of such metrics in SFP can also be important work to do. Finally, it is concluded in Section 3.4 that coupling metrics are evaluated using products developed in C, C++, Java, or COBOL. However, their performance in other extensively used programming languages, like Python, C#, and Visual basic are yet to be evaluated. Likewise, few famous scripting languages like PHP, JavaScript, etc. may also be employed.

Bibliography

[1] P. Ammann and J. Offutt, *Introduction to Software Testing*, 1st ed. New York, NY, USA: Cambridge University Press, 2008.

[2] B. Hailpern and P. Santhanam, "Software debugging, testing, and verification," *IBM Systems Journal*, vol. 41, no. 1, pp. 4–12, 2002.

[3] Undo, "Study: Software failures cost the enterprise software market $61b annually," *PRNewswire*, may 2020.

[4] M. N. Ann, *Ensuring Software Reliability.* CRC Press, 2018.

[5] S. A. Sherer, "Software fault prediction," *Journal of Systems and Software*, vol. 29, no. 2, pp. 97–105, 1995.

[6] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[7] G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," *Vietnam J. of Computer Science*, vol. 1, no. 2, p. 79–95, may 2014.

[8] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, April 2005.

[9] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, "Comparing negative binomial and recursive partitioning models for fault prediction," in *Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, ser. PROMISE '08. New York, NY, USA: ACM, 2008, pp. 3–10.

[10] S. Watanabe, H. Kaiya, and K. Kaijiri, "Adapting a fault prediction model to allow inter languagereuse," in *Proceedings of the 4th International Workshop*

on Predictor Models in Software Engineering, ser. PROMISE '08. New York, NY, USA: ACM, 2008, pp. 19–24.

[11] E. Weyuker, T. Ostrand, and R. M. Bell, "Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models," *Empirical Software Engineering*, vol. 13, pp. 539–559, 10 2008.

[12] S. S. Rathore and S. Kumar, "A study on software fault prediction techniques," *Artificial Intelligence Review*, vol. 51, no. 2, pp. 255–327, Feb 2019.

[13] K. Sandeep and S. R. Santosh, *Software Fault Prediction, A Road Map.* Singapore: Springer Singapore, 2018.

[14] N. Seliya and T. M. Khoshgoftaar, "Software quality estimation with limited fault data: a semi-supervised learning perspective," *Software Quality Journal*, vol. 15, no. 3, pp. 327–344, 2007.

[15] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, "A systematic review of fault prediction approaches used in software engineering," Technical Report Lero-TR-2010-04, Lero, Tech. Rep., 2010.

[16] C. Catal and B. Diri, "A systematic review of software fault prediction studies," *Expert Syst. Appl.*, vol. 36, no. 4, pp. 7346–7354, May 2009.

[17] S. M. Henry, "Information flow metrics for the evaluation of operating systems' structure," Ph.D. dissertation, Iowa State University, Ames, IA, USA, 1979, aAI8000138.

[18] S. M. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE Transactions on Software Engineering*, vol. 7, no. 5, pp. 510–518, Sep. 1981.

[19] D. A. Troy and S. H. Zweben, "Measuring the quality of structured designs," *Journal of Systems and Software*, vol. 2, no. 2, pp. 113–120, 1981.

[20] D. H. Hutchens and V. R. Basili, "System structure analysis: Clustering with data bindings," *IEEE transactions on Software Engineering*, no. 8, pp. 749–757, 1985.

[21] K. E. Emam, M. Walcelio, and C. M. Javam, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, 02 2001.

[22] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[23] K. K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study," *Software Process: Improvement and Practice*, vol. 14, no. 1, pp. 39–62, jan 2009.

[24] G. J. Myers, *Reliable Software Through Composite Design.* Petrocelli/Charter, 1975.

[25] M. Page-Jones, *The Practical Guide to Structured Systems Design: 2Nd Edition.* Upper Saddle River, NJ, USA: Yourdon Press, 1988.

[26] A. J. Offutt, M. J. Harrold, and P. Kolte, "A software metric system for module coupling," *Journal of Systems and Software*, vol. 20, no. 3, pp. 295–308, Mar. 1993.

[27] T. Lethbridge and R. Laganiere, *Object-Oriented Software Engineering: Practical Software Development using UML and Java.* McGraw-Hill Companies,Incorporated, 2002.

[28] S. Anwer, A. Adbellatif, M. Alshayeb, and M. S. Anjum, "Effect of coupling on software faults: An empirical study," in *2017 International Conference on Communication, Computing and Digital Systems (C-CODE)*, March 2017, pp. 211–215.

[29] K. Johari and A. Kaur, "Validation of object oriented metrics using open source software system: An empirical study," *ACM Sigsoft Software Engineering Notes*, vol. 37, pp. 1–4, 01 2012.

[30] R. Malhotra, A. Kaur, and Y. Singh, "Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines," *International Journal of System Assurance Engineering and Management*, vol. 1, pp. 269–281, 09 2010.

[31] R. Selvarani, T. R. G. Nair, and V. K. Prasad, "Estimation of defect proneness using design complexity measurements in object-oriented software," in *2009 International Conference on Signal Processing Systems*, May 2009, pp. 766–770.

[32] R. Shatnawi and W. Li, "The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process," *Journal of Systems and Software*, vol. 81, no. 11, pp. 1868–1882, Nov. 2008.

[33] B. Goel and Y. Singh, *Empirical Investigation of Metrics for Fault Prediction on Object-Oriented Software.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 255–265.

[34] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Investigating effect of design metrics on fault proneness in object-oriented systems." *Journal of Object Technology*, vol. 6, no. 10, pp. 127–141, 2007.

[35] Yuming Zhou and Hareton Leung, "Empirical analysis of object-oriented design metrics for predicting high and low severity faults," *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771–789, Oct 2006.

[36] R. Shatnawi, W. Li, and H. Zhang, "Predicting error probability in the eclipse project," in *Software Engineering Research and Practice*, 01 2006, pp. 422–428.

[37] P. Yu, T. Systa, and H. Müller, "Predicting fault-proneness using oo metrics. an industrial case study," 02 2002, pp. 99 – 107.

[38] L. Briand, J. Wüst, and H. Lounis, "Replicated case studies for investigating quality factors in object-oriented designs," *Empirical Software Engineering*, vol. 6, pp. 11–58, 03 2001.

[39] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *Journal of Systems and Software*, vol. 51, no. 3, pp. 245–273, 2000.

[40] K. El Emam, S. Benlarbi, N. Goel, and S. Rai, *A validation of object-oriented metrics.* National Research Council Canada, Institute for Information Technology, 1999.

[41] L. C. Briand, J. Daly, V. Porter, and J. Wust, "A comprehensive empirical validation of design measures for object-oriented systems," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No.98TB100262)*, Nov 1998, pp. 246–257.

[42] A. B. Binkley and S. R. Schach, "Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures," in

Proceedings of the 20th International Conference on Software Engineering, April 1998, pp. 452–455.

[43] A. B. Binkley and S. R. Schach, "Metrics for predicting run-time failures," Technical Report 97-03, Computer Science Department, Vanderbilt University, Nashville, NT, Tech. Rep., 1997.

[44] A. Binkley, S. Schach *et al.*, "Inheritance-based metrics for predicting maintenance effort: an empirical study," *Computer Science Department, Vanderbilt University, Tech. Rep. TR*, vol. 9705, 1997.

[45] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on Software Engineering*, vol. 22, no. 10, pp. 751–761, Oct 1996.

[46] D. Kumari and K. Rajnish, "Investigating the effect of object-oriented metrics on fault proneness using empirical analysis," *International Journal of Software Engineering and its Applications*, vol. 9, pp. 171–188, 01 2015.

[47] P. He, B. Li, X. Liu, J. Chen, and Y. Ma, "An empirical study on software defect prediction with a simplified metric set," *Information and Software Technology*, vol. 59, no. C, pp. 170–190, Mar. 2015.

[48] M. O. Elish, A. H. Al-Yafei, and M. Al-Mulhem, "Empirical comparison of three metrics suites for fault prediction in packages of object-oriented systems: A case study of eclipse," *Advances in Engineering Software*, vol. 42, no. 10, pp. 852–859, oct 2011.

[49] S. S. Rathore and A. Gupta, "Validating the effectiveness of object-oriented metrics over multiple releases for predicting fault proneness," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2012, pp. 350–355.

[50] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[51] R. Shatnawi, "A quantitative investigation of the acceptable risk levels of object-oriented metrics in open-source systems," *IEEE Transactions on Software Engineering*, vol. 36, no. 2, pp. 216–225, March 2010.

[52] S. Kpodjedo, F. Ricca, G. Antoniol, and P. Galinier, "Evolution and search based metrics to improve defects prediction," in *2009 1st International Symposium on Search Based Software Engineering*, May 2009, pp. 23–32.

[53] M. English, C. Exton, I. Rigon, and B. Cleary, "Fault detection and prediction in an open-source software project," in *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, ser. PROMISE '09. New York, NY, USA: ACM, 2009, pp. 17:1–17:11.

[54] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *Proceedings of the 30th International Conference on Software Engineering*, ser. ICSE '08. New York, NY, USA: ACM, 2008, pp. 531–540.

[55] J. Xu, D. Ho, and L. F. Capretz, "An empirical validation of object-oriented design metrics for fault prediction," vol. 4, 2008, pp. 571–577.

[56] G. J. Pai and J. Bechta Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on Software Engineering*, vol. 33, no. 10, pp. 675–686, Oct 2007.

[57] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. 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 Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, June 2007.

[58] K. El Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics," *IEEE Transactions on Software Engineering*, vol. 27, no. 7, pp. 630–650, July 2001.

[59] Mei-Huei Tang, Ming-Hung Kao, and Mei-Hwa Chen, "An empirical study on object-oriented metrics," in *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*, Nov 1999, pp. 242–249.

[60] G. Antoniol, R. Fiutem, and L. Cristoforetti, "Using metrics to identify design patterns in object-oriented software," in *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International.* IEEE, 1998, pp. 23–34.

[61] L. C. Briand, J. Wust, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *Software Maintenance,*

1999.(ICSM'99) Proceedings. IEEE International Conference on. IEEE, 1999, pp. 475–482.

[62] F. B. E Abreu, G. Pereira, and P. Sousa, "A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems," in *Software Maintenance and Reengineering, 2000. Proceedings of the Fourth European.* IEEE, 2000, pp. 13–22.

[63] A. Akaikine, "The impact of software design structure on product maintenance costs and measurement of economic benefits of product redesign," Ph.D. dissertation, Massachusetts Institute of Technology, 2010.

[64] D. J. Sturtevant, "System design and the cost of architectural complexity," Ph.D. dissertation, Massachusetts Institute of Technology, 2013.

[65] R. Lagerström, C. Baldwin, A. MacCormack, D. Sturtevant, and L. Doolan, "Exploring the relationship between architecture coupling and software vulnerabilities," in *International Symposium on Engineering Secure Software and Systems.* Springer, 2017, pp. 53–69.

[66] G. Gui and P. D. Scott, "New coupling and cohesion metrics for evaluation of software component reusability," in *2008 The 9th International Conference for Young Computer Scientists*, Nov 2008, pp. 1181–1186.

[67] G. Gui and P. D. Scott, "Ranking reusability of software components using coupling metrics," *Journal of Systems and Software*, vol. 80, no. 9, pp. 1450–1459, Sep. 2007.

[68] D. Hristov, O. Hummel, M. Huq, and W. Janjic, "Structuring software reusability metrics for component-based software development," in *Proceedings of Int. Conference on Software Engineering Advances (ICSEA)*, vol. 226, 2012.

[69] P. K. Bhatia and R. Mann, "An approach to measure software reusability of oo design," in *Proceedings of the 2nd National Conference on Challenges & Opportunities in Information Technology.* Citeseer, 2008, pp. 26–30.

[70] P. Trivedi and R. Kumar, "Software metrics to estimate software quality using software component reusability," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 2, p. 144, 2012.

[71] F. G. Wilkie and B. A. Kitchenham, "An investigation of coupling, reuse and maintenance in a commercial c++ application," *Information and Software Technology*, vol. 43, no. 13, pp. 801–812, 2001.

[72] J. Sanz-Rodriguez, J. M. Dodero, and S. Sanchez-Alonso, "Metrics-based evaluation of learning object reusability," *Software Quality Journal*, vol. 19, no. 1, pp. 121–140, 2011.

[73] G. Gui and P. D. Scott, "Measuring software component reusability by coupling and cohesion metrics," *Journal of Computers*, vol. 4, pp. 797–805, 2009.

[74] A. Kumar, "Measuring software reusability using svm based classifier approach," *International Journal of Information Technology and Knowledge Management*, vol. 5, no. 1, pp. 205–209, 2012.

[75] A. Shri, P. S. Sandhu, V. Gupta, and S. Anand, "Prediction of reusability of object oriented software systems using clustering approach," *World academy of science, Engineering and Technology*, vol. 43, pp. 853–856, 2010.

[76] F. Dandashi, "A method for assessing the reusability of object-oriented code using a validated set of automated measurements," in *Proceedings of the 2002 ACM symposium on Applied computing.* ACM, 2002, pp. 997–1003.

[77] G. Gui and P. D. Scott, "Coupling and cohesion measures for evaluation of component reusability," in *Proceedings of the 2006 international workshop on Mining software repositories.* ACM, 2006, pp. 18–21.

[78] L. H. Etzkorn, W. E. Hughes Jr, and C. G. Davis, "Automated reusability quality analysis of oo legacy software," *Information and Software Technology*, vol. 43, no. 5, pp. 295–308, 2001.

[79] J. C. C. P. Mascena, E. S. de Almeida, and S. R. de Lemos Meira, "A comparative study on software reuse metrics and economic models from a traceability perspective," in *IRI-2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.* IEEE, 2005, pp. 72–77.

[80] H. Li and B. Li, "A pair of coupling metrics for software networks," *Journal of Systems Science and Complexity*, vol. 24, no. 1, pp. 51–60, 2011.

[81] M. Ajmal, H. Kabaili, R. K Keller, F. Lustman, and G. Saint-Denis, "Design properties and object-oriented software changeability," *Proceedings of the*

Euromicro Conference on Software Maintenance and Reengineering, CSMR, 01 2000.

[82] S. Rongviriyapanish, T. Wisuttikul, B. Charoendouysil, P. Pitakket, P. Anancharoenpakorn, and P. Meananeatra, "Changeability prediction model for java class based on multiple layer perceptron neural network," in *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. IEEE, 2016, pp. 1–6.

[83] A. Parashar and J. Chhabra, "Mining software change data stream to predict changeability of classes of object-oriented software system," *Evolving Systems*, vol. 7, 04 2016.

[84] Z. Rosko, "Predicting the changeability of software product lines for business application," in *ISD*, 09 2014.

[85] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the probability of change in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 31, pp. 601– 614, 08 2005.

[86] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *International Journal of Machine Learning and Cybernetics*, vol. 4, 08 2012.

[87] L. Kumar, S. K. Rath, and A. Sureka, "Empirical analysis on effectiveness of source code metrics for predicting change-proneness," in *Proceedings of the 10th Innovations in Software Engineering Conference*, ser. ISEC '17. New York, NY, USA: ACM, 2017, pp. 4–14.

[88] X. Sun, H. Leung, L. Bin, and B. Li, "Change impact analysis and changeability assessment for achange proposal: An empirical study ** **," *Journal of Systems and Software*, vol. 96, 10 2014.

[89] L. C. Briand, J. Wust, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*, Aug 1999, pp. 475–482.

[90] M. K. Abdi, H. Lounis, and H. Sahraoui, "Analyzing change impact in object-oriented systems," in *32nd EUROMICRO Conference on Software*

Engineering and Advanced Applications (EUROMICRO'06), Aug 2006, pp. 310–319.

[91] E. Arisholm, "Empirical assessment of the impact of structural properties on the changeability of object-oriented software," *Information and Software Technology*, vol. 48, pp. 1046–1055, 11 2006.

[92] A.-R. Han, S.-U. Jeon, D.-H. Bae, and J.-E. Hoing, "Measuring behavioral dependency for improving change-proneness prediction in uml-based design models," *Journal of Systems and Software*, vol. 83, pp. 222–234, 02 2010.

[93] M. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, 05 2013.

[94] Y. Ayalew and K. Mguni, "An assessment of changeability of open source software," *Computer and Information Science*, vol. 6, pp. 68–79, 05 2013.

[95] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, March 2011, pp. 566–571.

[96] M. Abdi, H. Lounis, and H. Sahraoui, "Predicting change impact in object-oriented applications with bayesian networks," *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 1, pp. 234–239, 07 2009.

[97] L. Rosenberg, T. Hammer, and J. Shaw, "Software metrics and reliability," in *9th international symposium on software reliability engineering*, 1998.

[98] N. Nagappan, L. Williams, and M. Vouk, "Towards a metric suite for early software reliability assessment," in *International Symposium on Software Reliability Engineering, FastAbstract, Denver, CO*, 2003, pp. 238–239.

[99] N. Nagappan, "A software testing and reliability early warning (strew) metric suite," Ph.D. dissertation, 2005, aAI3162465.

[100] H. Okamura, Y. Etani, and T. Dohi, "A multi-factor software reliability model based on logistic regression," in *Proceedings of the 2010 IEEE 21st International Symposium on Software Reliability Engineering*, ser. ISSRE '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 31–40.

[101] M. Li, C. Smidts, and R. W. Brill, “Ranking software engineering measures related to reliability using expert opinion,” in *Proceedings 11th International Symposium on Software Reliability Engineering. ISSRE 2000*, Oct 2000, pp. 246–258.

[102] L. Guo, “Software quality and reliability prediction using dempster-shafer theory,” Ph.D. dissertation, Morgantown, WV, USA, 2004, aAI3156448.

[103] I. Chowdhury and M. Zulkernine, “Can complexity, coupling, and cohesion metrics be used as early indicators of vulnerabilities?” in *Proceedings of the 2010 ACM Symposium on Applied Computing.* ACM, 2010, pp. 1963–1969.

[104] A. Yadav and R. Khan, “Impact of cohesion on reliability,” *Journal of Information and operations Management*, vol. 3, no. 1, p. 191, 2012.

[105] C. Rajaraman and M. R. Lyu, “Reliability and maintainability related software coupling metrics in c++ programs,” in *[1992] Proceedings Third International Symposium on Software Reliability Engineering*, Oct 1992, pp. 303–311.

[106] W. Li and S. M. Henry, “Object-oriented metrics that predict maintainability,” *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, Nov. 1993.

[107] M. Dagpinar and J. H. Jahnke, “Predicting maintainability with object-oriented metrics - an empirical comparison,” in *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, Nov 2003, pp. 155–164.

[108] Y. Lee and K. H. Chang, “Reusability and maintainability metrics for object-oriented software,” in *Proceedings of the 38th Annual on Southeast Regional Conference*, ser. ACM-SE 38. New York, NY, USA: ACM, 2000, pp. 88–94.

[109] P. Oman and J. Hagemeister, “Metrics for assessing a software system’s maintainability,” in *Proceedings Conference on Software Maintenance 1992*, Nov 1992, pp. 337–344.

[110] Y. Zhou and B. Xu, “Predicting the maintainability of open source software using design metrics,” *Wuhan University Journal of Natural Sciences*, vol. 13, pp. 14–20, 02 2008.

[111] P. Bengtsson, “Towards maintainability metrics on software architecture: An adaptation of object-oriented metrics,” in *First Nordic Workshop on Software Architecture (NOSA'98)*, 1998.

[112] J. Al Dallal, “Object-oriented class maintainability prediction using internal quality attributes,” *Information and Software Technology*, vol. 55, no. 11, pp. 2028–2048, 2013.

[113] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, “Application of artificial neural network for predicting maintainability using object-oriented metrics,” *Transactions on Engineering, Computing, and Technology*, vol. 15, pp. 285–289, 2006.

[114] R. K. Bandi, V. K. Vaishnavi, and D. E. Turk, “Predicting maintenance performance using object-oriented design complexity metrics,” *IEEE Transactions on Software Engineering*, vol. 29, no. 1, pp. 77–87, Jan 2003.

[115] M. Frappier, S. Matwin, and A. Mili, “Software metrics for predicting maintainability,” *Software Metrics Study: Technical Memorandum 2*, vol. 2, 1994.

[116] H. Washizaki, T. Nakagawa, Y. Saito, and Y. Fukazawa, “A coupling-based complexity metric for remote component-based software systems toward maintainability estimation,” in *2006 13th Asia Pacific Software Engineering Conference (APSEC'06)*, Dec 2006, pp. 79–86.

[117] A. B. Binkley and S. R. Schach, “Prediction of run-time failures using static product quality metrics,” *Software Quality Journal*, vol. 7, no. 2, pp. 141–147, Jul 1998.

[118] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.

[119] S. McConnell, *Code complete.* Pearson Education, 2004.

[120] M. J. Foley, “Bugfest! win2000 has 63,000 'defects',” Feb 2000.

[121] G. J. Myers, *The Art of Software Testing.* John Wiley & Sons, 1979.

[122] I. Gondra, “Applying machine learning to software fault-proneness prediction,” *Journal of Systems and Software*, vol. 81, no. 2, pp. 186–195, 2008.

[123] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction,” *Applied Soft Computing*, vol. 27, no. C, pp. 504–518, Feb. 2015.

[124] T. Chappelly, C. Cifuentes, P. Krishnan, and S. Gevay, “Machine learning for finding bugs: An initial report,” in *Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE), IEEE Workshop on.* IEEE, 2017, pp. 21–26.

[125] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, “Heterogeneous defect prediction,” *IEEE Transactions on Software Engineering*, 2017.

[126] Z. Li, X.-Y. Jing, and X. Zhu, “Progress on approaches to software defect prediction,” *Institution of Engineering and Technology Software*, vol. 12, no. 3, pp. 161–175, 2018.

[127] T. J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308–320, Jul. 1976.

[128] M. H. Halstead, *Elements of Software Science (Operating and Programming Systems Series).* New York, NY, USA: Elsevier Science Inc., 1977.

[129] K. L. Morris, “Metrics for object-oriented software development environments,” PhD dissertation, Massachusetts Institute of Technology, 1989.

[130] S. R. Chidamber and C. F. , “Towards a metrics suite for object oriented design,” *Special Interest Group on Programming Languages*, vol. 26, no. 11, pp. 197–211, Nov. 1991.

[131] P. Oman, J. Hagemeister, and D. Ash, “A definition and taxonomy for software maintainability, report setl report 91-08-tr,” *University of Idaho*, 1991.

[132] J. Chen and J. Lu, “A new metric for object-oriented design,” *Information and software technology*, vol. 35, no. 4, pp. 232–240, 1993.

[133] Y. Lee, B. Liang, S. Wu, and F. Wang, “Measuring the coupling and cohesion of an object-oriented program based on information flow,” in *Proc. International Conference on Software Quality, Maribor, Slovenia*, 1995, pp. 81–90.

[134] F. B. Abreu, M. Goulão, and R. Esteves, "Toward the design quality evaluation of object-oriented software systems," in *Proceedings of the 5th International Conference on Software Quality, Austin, Texas, USA*, 1995, pp. 44–57.

[135] R. D. Martin, "Object oriented design quality metrics: an analysis of dependencies," vol. 2, no. 3, 1995.

[136] M. Lorenz and J. Kidd, *Object-oriented Software Metrics: A Practical Guide.* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1994.

[137] W. Li, "Another metric suite for object-oriented programming," *Journal of Systems and Software*, vol. 44, no. 2, pp. 155–162, Dec. 1998.

[138] L. Son, N. Pritam, M. Khari, R. Kumar, P. Phuong, and T. Pham, "Empirical study of software defect prediction: A systematic mapping," *Symmetry*, vol. 11, p. 212, 02 2019.

[139] G. Boetticher, T. Menzies, and T. Ostrand, "{PROMISE} repository of empirical software engineering data," 01 2007.

[140] M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data quality: Some comments on the nasa software defect datasets," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1208–1215, Sept 2013.

[141] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, vol. 14, no. 5, pp. 540–578, Oct 2009.

[142] M. D'Ambros, M. Lanza, and R. Robbes, "An extensive comparison of bug prediction approaches," in *Proceedings of MSR 2010 (7th IEEE Working Conference on Mining Software Repositories).* IEEE CS Press, 2010, pp. 31 – 41.

[143] R. wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," 09 2011, pp. 15–25.

[144] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, 2010, pp. 9:1–9:10.

[145] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, ser. PROMISE ’07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 9–.

[146] G. Mauša, T. G. Grbac, and B. D. Bašić, “Data collection for software defect prediction - an exploratory case study of open source software projects,” in *2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, 2015, pp. 463–469.

[147] A. Janes, M. Scotto, W. Pedrycz, B. Russo, M. Stefanovic, and G. Succi, “Identification of defect-prone classes in telecommunication software systems using design metrics,” *Information Sciences*, vol. 176, no. 24, pp. 3711–3734, Dec. 2006.

[148] R. Jabangwe, J. Börstler, D. Šmite, and C. Wohlin, “Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review,” *Empirical Software Engineering*, vol. 20, no. 3, pp. 640–693, Jun 2015.

[149] O. A. Qasem, M. Akour, and M. Alenezi, “The influence of deep learning algorithms factors in software fault prediction,” *IEEE Access*, vol. 8, pp. 63 945–63 960, 2020.

[150] A. Haveri and Y. Suresh, “Software fault prediction using artificial intelligence techniques,” in *2017 2nd International Conference on Computational Systems and Information Technology for Sustainable Solution (CSITSS)*, 2017, pp. 1–5.

[151] A. Okutan, “Use of source code similarity metrics in software defect prediction,” *CoRR*, vol. abs/1808.10033, 2018.

[152] Y. Shao, B. Liu, S. Wang, and G. li, “A novel software defect prediction based on atomic class-association rule mining,” *Expert Systems with Applications*, vol. 114, pp. 237–254, 07 2018.

[153] Qinbao Song, M. Shepperd, M. Cartwright, and C. Mair, “Software defect association mining and defect correction effort prediction,” *IEEE Transactions on Software Engineering*, vol. 32, no. 2, pp. 69–82, 2006.

[154] R. Karthik and N. Manikandan, “Defect association and complexity prediction by mining association and clustering rules,” in *2010 2nd International*

Conference on Computer Engineering and Technology, vol. 7, 2010, pp. V7–569–V7–573.

[155] C.-P. Chang, C.-P. Chu, and Y.-F. Yeh, “Integrating in-process software defect prediction with association mining to discover defect pattern,” *Information and Software Technology*, vol. 51, pp. 375–384, 02 2009.

[156] M. K. Thota, F. H. Shajin, P. Rajesh *et al.*, “Survey on software defect prediction techniques,” *International Journal of Applied Science and Engineering*, vol. 17, no. 4, pp. 331–344, 2020.

[157] L. Guo, Y. Ma, B. Cukic, and Harshinder Singh, “Robust prediction of fault-proneness by random forests,” in *15th International Symposium on Software Reliability Engineering*, Nov 2004, pp. 417–428.

[158] A. Kaur and R. Malhotra, “Application of random forest in predicting fault-prone classes,” in *2008 International Conference on Advanced Computer Theory and Engineering*, Dec 2008, pp. 37–43.

[159] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, “Benchmarking classification models for software defect prediction: A proposed framework and novel findings,” *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, July 2008.

[160] Y. Ma, L. Guo, B. Cukic, and Lane, “A statistical framework for the prediction of fault-proneness,” *Advances in Machine Learning Applications in Software Engineering*, 01 2006.

[161] Y. Jiang, B. Cukic, and T. Menzies, “Fault prediction using early lifecycle data,” in *The 18th IEEE International Symposium on Software Reliability (ISSRE ’07)*, Nov 2007, pp. 237–246.

[162] C. Catal and B. Diri, “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem,” *Information Sciences*, vol. 179, no. 8, pp. 1040–1058, Mar. 2009.

[163] Y. Zhou, B. Xu, and H. Leung, “On the ability of complexity metrics to predict fault-prone classes in object-oriented systems,” *Journal of Systems and Software*, vol. 83, pp. 660–674, 04 2010.

[164] N. Chinchor, “Muc-4 evaluation metrics,” in *Proceedings of the 4th Conference on Message Understanding*, ser. MUC4 ’92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 22–29.

[165] Y. Wand and R. Weber, "An ontological model of an information system," *IEEE Transactions on Software Engineering*, vol. 16, no. 11, pp. 1282–1292, Nov. 1990.

[166] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, p. 117, Jun. 1974.

[167] P. J. Kaur and S. Kaushal, "Cohesion and coupling measures for aspect oriented systems," AETS/7/590, Elsevier, Tech. Rep., 2013.

[168] E. Arisholm, L. C. Briand, and A. Foyen, "Dynamic coupling measurement for object-oriented software," *IEEE Transactions on Software Engineering*, vol. 30, no. 8, pp. 491–506, 2004.

[169] E. Fregnan, T. Baum, F. Palomba, and A. Bacchelli, "A survey on software coupling relations and tools," *Information and Software Technology*, vol. 107, 11 2018.

[170] D. Poshyvanyk and A. Marcus, "The conceptual coupling metrics for object-oriented systems," in *2006 22nd IEEE International Conference on Software Maintenance*, Sep. 2006, pp. 469–478.

[171] F. Tsui, O. Karam, and B. Bernal, *Essentials of Software Engineering.* Jones & Bartlett Learning, 2016.

[172] E. Braude and M. Bernstein, *Software Engineering: Modern Approaches, Second Edition.* New York, NY, USA: John Wiley, Mar 2016.

[173] P. Laplante, *Software Engineering for Image Processing Systems*, ser. Image processing series. CRC Press, 2015.

[174] C. Ebert and I. Morschel, "Metrics for quality analysis and improvement of object-oriented software," *Information and Software Technology*, vol. 39, no. 7, pp. 497–509, 1997.

[175] T. P. Johnson, *Snowball Sampling.* American Cancer Society, 2005.

[176] N. E. Fenton and M. Neil, "A critique of software defect prediction models," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, pp. 675–689, Sep. 1999.

[177] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, pp. 1040–1058, 03 2009.

[178] E. Arisholm, L. Briand, and E. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, pp. 2–17, 01 2010.

[179] B. Kitchenham, "What's up with software metrics? - a preliminary mapping study," *Journal of Systems and Software*, vol. 83, no. 1, pp. 37–51, Jan. 2010.

[180] S. Beecham, T. Hall, D. Bowes, D. Gray, S. Counsell, and S. Black, "A systematic review of fault prediction approaches used in software engineering," *The Irish Software Engineering Research Centre: Limerick, Ireland*, 2010.

[181] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[182] R. S. Wahono, "A systematic literature review of software defect prediction: research trends, datasets, methods and frameworks," *Journal of Software Engineering*, vol. 1, no. 1, pp. 1–16, 2015.

[183] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, Nov 2012.

[184] B. Isong and E. Obeten, "A systematic review of the empirical validation of object-oriented metrics towards fault-proneness prediction," *International Journal of Software Engineering and Knowledge Engineering*, vol. 23, no. 10, pp. 1513–1540, 2013.

[185] S. Singh and A. G. Singh, "Enhancing object oriented coupling metrics wrt connectivity patterns," Ph.D. dissertation, Thapar university, Patialay, Thapar university, Patialay, 2014, 147004.

[186] J. Eder, G. Kappel, and M. Schrefl, "Coupling and cohesion in object-oriented systems," Citeseer, Tech. Rep., 1994.

[187] M. Lanza and R. Marinescu, *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*, 01 2006.

[188] D. N. Card and W. W. Aggresti, "Measuring software design complexity," *Journal of Systems and Software*, vol. 8, no. 3, pp. 185–197, Jun. 1988.

[189] L. M. Laird and M. C. Brennan, *Software measurement and estimation: a practical approach.* John Wiley & Sons, 2006, vol. 2.

[190] T. J. McCabe and C. W. Butler, "Design complexity measurement and testing," *Communications of the ACM*, vol. 32, no. 12, pp. 1415–1425, Dec. 1989.

[191] N. Fenton and A. Melton, "Deriving structurally based software measures," *Journal of Systems and Software*, vol. 12, no. 3, pp. 177–187, Jul. 1990.

[192] R. Harrison, S. Counsell, and R. Nithi, "Coupling metrics for object-oriented design," in *Proceedings Fifth International Software Metrics Symposium. Metrics (Cat. No. 98TB100262).* IEEE, 1998, pp. 150–157.

[193] H. A. Sahraoui, R. Godin, and T. Miceli, "Can metrics help to bridge the gap between the improvement of oo design quality and its automation?" in *Proceedings of the International Conference on Software Maintenance (ICSM'00)*, ser. ICSM '00. Washington, DC, USA: IEEE Computer Society, 2000, pp. 154–.

[194] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[195] S. A. Miquirice and R. S. Wazlawick, "Relationship between cohesion and coupling metrics for object-oriented systems," in *Information and Software Technologies*, R. Damaševičius and G. Vasiljevienė, Eds. Cham: Springer International Publishing, 2018, pp. 424–436.

[196] F. B. Abreu and R. Carapuça, "Object-oriented software engineering: Measuring and controlling the development process," in *Proceedings of the 4th international conference on software quality*, vol. 186, 1994, pp. 1–8.

[197] H. Dhama, "Quantitative models of cohesion and coupling in software," *Journal of Systems and Software*, vol. 29, no. 1, pp. 65–74, Apr. 1995.

[198] A. B. Binkley and S. R. Schach, "Toward a unified approach to coupling," in *Proceedings of the 35th Annual Southeast Regional Conference*, ser. ACM-SE 35. New York, NY, USA: ACM, 1997, pp. 91–97.

[199] L. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for c++," in *Proceedings of the (19th) International Conference on Software Engineering*, May 1997, pp. 412–421.

[200] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. E. Lorensen *et al.*, *Object-oriented modeling and design.* Prentice-hall Englewood Cliffs, NJ, 1991, vol. 199, no. 1.

[201] J. Fitzpatrick, "More c++ gems," R. C. Martin, Ed. New York, NY, USA: Cambridge University Press, 2000, ch. Applying the ABC Metric to C, C++, and Java, pp. 245–264.

[202] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, Jan 2002.

[203] J. Shao and Y. Wang, "A new measure of software complexity based on cognitive weights," in *CCECE 2003 - Canadian Conference on Electrical and Computer Engineering. Toward a Caring and Humane Technology (Cat. No.03CH37436)*, vol. 2, May 2003, pp. 1333–1338 vol.2.

[204] N. Nachiappan and B. Thirumalesh, "Technologies for code failure proneness estimation," *Computer Science Department, Vanderbilt University, Tech. Rep. TR*, 26 April 2007.

[205] J. S. Alghamdi, "Measuring software coupling," *Arabian Journal for Science and Engineering*, vol. 33, no. 1, p. 119, 2008.

[206] B. A. Kitchenham, L. M. Pickard, and S. J. Linkman, "An evaluation of some design metrics," *Software Engineering Journal*, vol. 5, no. 1, p. 50–58, Jan. 1990.

[207] R. Subramanyam and M. S. Krishnan, "Empirical analysis of ck metrics for object-oriented design complexity: implications for software defects," *IEEE Transactions on Software Engineering*, vol. 29, no. 4, pp. 297–310, April 2003.

[208] A. Abubakar, J. AlGhamdi, and M. Ahmed, "Can cohesion predict fault density?" in *IEEE International Conference on Computer Systems and Applications, 2006*, March 2006, pp. 890–893.

[209] Kumar, A. Tirkey, and S. Rath, "An effective fault prediction model developed using an extreme learning machine with various kernel methods," *Frontiers of Information Technology and Electronic Engineering*, vol. 19, pp. 864–888, 07 2018.

[210] B. Ndemeye, S. Hussain, and B. Norris, "Threshold-based analysis of the code quality of high-performance computing software packages."

[211] D. Sharma and P. Chandra, "An empirical analysis of software fault proneness using factor analysis with regression," 2021.

[212] R. Malhotra and J. Jain, "Predicting defects in object-oriented software using cost-sensitive classification," *IOP Conference Series: Materials Science and Engineering*, vol. 1022, p. 012112, 01 2021.

[213] N. Kaur and H. Singh, "An empirical assessment of threshold techniques to discriminate the fault status of software," *Journal of King Saud University - Computer and Information Sciences*, 03 2021.

[214] S. Marangoz, B. Mutlu, and E. A. Sezer, "Fuzzy cognitive maps for software fault prediction," in *2021 15th Turkish National Software Engineering Symposium (UYMS)*, 2021, pp. 1–6.

[215] S. K. Pandey and A. K. Tripathi, "Class imbalance issue in software defect prediction models by various machine learning techniques: An empirical study," in *2021 8th International Conference on Smart Computing and Communications (ICSCC)*, 2021, pp. 58–63.

[216] R. C. Martin, *Agile software development: principles, patterns, and practices.* Prentice Hall, 2002.

[217] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide.* Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., 1999.

[218] L. Yu, S. R. Schach, K. Chen, and J. Offutt, "Categorization of common coupling and its application to the maintainability of the linux kernel," *IEEE Transactions on Software Engineering*, vol. 30, no. 10, pp. 694–706, 2004.

[219] J. AlGhamdi, M. Elish, and M. Ahmed, "A tool for measuring inheritance coupling in object-oriented systems," *information SCiences*, vol. 140, no. 3-4, pp. 217–227, 2002.

[220] N. Kayarvizhy and S. Kanmani, "An automated tool for computing object oriented metrics using xml," in *Advances in Computing and Communications*, A. Abraham, J. Lloret Mauri, J. F. Buford, J. Suzuki, and S. M. Thampi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 69–79.

[221] R. C. Gronback, “Software remodeling : Improving design and implementation quality using audits , metrics and refactoring in borland,” 2003.

[222] D. Spinellis, “Tool writing: a forgotten art? (software tools),” *IEEE Software*, vol. 22, no. 4, pp. 9–11, 2005.

[223] M. Wilhelm and S. Diehl, “Dependency viewer - a tool for visualizing package design quality metrics,” in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–2.

[224] J. Offutt, A. Abdurazik, and S. R. Schach, “Quantitatively measuring object-oriented couplings,” *Software Quality Journal*, vol. 16, no. 4, p. 489–512, Dec. 2008.

[225] P. Rosner and S. Viswanathan, “Visualization of coupling and programming to interface for object-oriented systems,” in *2008 12th International Conference Information Visualisation*, 2008, pp. 575–581.

[226] V. S. Bidve and P. Sarasu, “Tool for measuring coupling in object-oriented java software,” *International Journal of Engineering and Technology*, vol. 8, no. 2, pp. 812–820, 2016.

[227] S. T. Inc., “Scitools maintenance, metrics and documentation tools for ada, c, c++, java and fortran,” Mar 2020.

[228] F. Zhang, Q. Zheng, Y. Zou, and A. E. Hassan, “Cross-project defect prediction using a connectivity-based unsupervised classifier,” in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, May 2016, pp. 309–320.

[229] N. Anquetil, “Predicting software defects with causality tests,” 2013.

[230] M. Tufano, C. Watson, G. Bavota, M. Di Penta, M. White, and D. Poshyvanyk, “Learning how to mutate source code from bug-fixes,” in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2018, pp. 301–312.

[231] ——, “An empirical investigation into learning bug-fixing patches in the wild via neural machine translation,” in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 832–837.

[232] W. H. J. David and L. a. X. S. Stanley, “Applied logistic regression.” Wiley, New York, 1989.

[233] E. Arisholm, L. C. Briand, and E. B. Johannessen, “A systematic and comprehensive investigation of methods to build and evaluate fault prediction models,” *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[234] E. J. Weyuker, T. J. Ostrand, and R. M. Bell, “Comparing the effectiveness of several modeling methods for fault prediction,” *Empirical Software Engineering*, vol. 15, no. 3, pp. 277–295, 2010.

[235] C. X. Ling, J. Huang, H. Zhang *et al.*, “AUC: A statistically consistent and more discriminating measure than accuracy,” in *IJCAI*, vol. 3, 2003, pp. 519–524.

[236] S. Rosset, “Model selection via the AUC,” in *Proceedings of the twenty-first international conference on Machine learning.* ACM, 2004, p. 89.

[237] J. Huang and C. X. Ling, “Using AUC and accuracy in evaluating learning algorithms,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[238] F. Wilcoxon, “Individual comparisons by ranking methods,” vol. 1, pp. 80–83, Dec 1945.

[239] A. Luque, A. Carrasco, A. Martín, and A. de las Heras, “The impact of class imbalance in classification performance metrics based on the binary confusion matrix,” *Pattern Recognition*, vol. 91, pp. 216 – 231, 2019.