MEASURING STABILITY OF OBJECT-ORIENTED SOFTWARE PACKAGES

BY

JAWAD JAVED AKBAR BAIG

A Thesis Presented to the

DEANSHIP OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

SOFTWARE ENGINEERING

DECEMBER 2017

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN 31261, SAUDI ARABIA

DEANSHIP OF GRADUATE STUDIES

This thesis, written by **JAWAD JAVED AKBAR BAIG** under the direction of his thesis adviser and approved by his thesis committee, has been presented to and accepted by the Dean of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE IN SOFTWARE ENGINEERING**.

Thesis Committee

Sajjad Mahmood

Dr. Sajjad Mahmood (Adviser)

Alshayeb

Dr. Mohammad Alshayeb (Member)

Dr. Mahmood Niazi (Member)

Dr. Khalid Al-Jasser
Department Chairman

Dr. Salam A. Zummo
Dean of Graduate Studies

15/2/2018
Date

Dedication

To my parents for their love and continues support.

ACKNOWLEDGMENTS

In the name of Allah, the Most Gracious, the Most Merciful.

First and foremost, Alhamdulillah All praise to Almighty Allah, who gave me the power to accomplish my master's degree.

I acknowledge King Fahd University of Petroleum & Minerals for supporting this research.

All appreciation to my advisor; Dr. Sajjad Mahmood, who helped me and encouraged me during my thesis journey; he was a teacher and a friend. I wish to thank my dissertation committee members, Dr. Mohammad Alshayeb and Dr. Mahmood Niazi, for their help and support.

Finally, I wish to express my gratitude to my family members for their continues support, patience and prayers. I would like to thank all my KFUPM colleagues, who provided me the encouragement dealing with difficult times during my thesis journey.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PSM	Package Stability Metrics
PCS	Package Content Stability
IPIS	Intra-Package Interaction Stability
EPIS	Inter-Package Interaction Stability
LOC	Line of Code
IPIS	Intra-Package Interaction Stability
CSM	Class Stability Metric
CII	Class Implementation Instability
SII	System Implementation Instability
SDI	System Design Instability
CDI	Core Design Instability
CCI	Core Call Instability
ASM	Architecture Stability Metric
RBSM	Relationship Based Similarity Metric
PCA	Principal Component Analysis
SVM	Support Vector Machine

THESIS ABSTRACT

NAME: JAWAD JAVED AKBAR BAIG

TITLE OF STUDY: MEASURING STABILITY OF OBJECT-ORIENTED SOFTWARE PACKAGES

MAJOR FIELD: SOFTWARE ENGINEERING

DATE OF DEGREE: December 2017

Software stability is an important object oriented design characteristic that contributes towards maintainability quality attribute. Software stability quantifies the sensitivity to change of a given system between different versions. Stable software tends to reduce the maintenance effort. Assessing software stability during the object oriented design phase is one of the measures to obtain maintainable software. To determine software stability, there are several metrics at the system and class levels, but at the package level, such metrics rarely exist. In this thesis, we propose a new Package Stability Metrics (PSM) based on the notion of changes between package contents, intra-package connections and inter-package connections. We validate the PSM theoretically and empirically. The theoretical validation is based on study of the mathematical properties of the metrics. The empirical validation

is carried out using four versions of five open source softwares and we also present a comparison with six comparable existing stability metrics. For empirical validation we present four analysis: (1) first analysis explores the correlation between package stability metrics and maintenance effort; (2) second analysis explores the correlations among six existing stability metrics and proposed package stability metrics; (3) third analysis applies principal component analysis to provide evidence that the new metrics captures new dimension of package stability; and (4) forth analysis applies linear regression analysis for maintenance effort prediction. The results show that PSM metrics suite provides better indication of package stability than existing stability metrics and is negatively correlated with maintenance effort. The analysis also proves that PSM metrics suite cover new dimension of package stability and increase the prediction accuracy of maintenance effort. We also presents the performance of different regression algorithms and classification algorithms to predict package maintainability.

ملخص الرسالة

الاسم الكامل: جواد جاويد اكبر بيک

عنوان الرسالة: قياس استقرار حزم البرامج الموجه الهدف

التخصص: هندسة برمجيات

تاريخ الدرجة العلمية: ديسمبر 2017

إن استقرار البرمجيات هو سمة مهمة وموجهة نحو تصميم الكائن الذي يساهم في صفة جودة الصيانة. استقرار البرمجيات يحدد حساسية تغيير نظام معين بين الإصدارات المختلفة .تميل البرامج الثابتة إلى تقليل جهد الصيانة .إن تقييم استقرار البرمجيات خلال مرحلة التصميم الموجه الكائن هو واحد من التدابير للحصول على البرمجيات القابلة للصيانة .ولتحديد استقرار البرنامج، هناك عدة مقاييس على مستوى النظام ومستوى الصف، ولكن على مستوى الحزمة، نادرا ما توجد هذه المقاييس .في هذه الأطروحة، نقترح مقاييس استقرار الحزمة الجديدة PSM استنادا إلى فكرة التغييرات بين محتويات الحزمة، والاتصالات داخل الحزمة والاتصالات بين الحزمة. قمنا بالتحقق من صحة PSM نظريا وتجريبيا. ويستند التحقق النظري إلى دراسة الخصائص الرياضية للمقاييس. يتم التحقق التجريبي باستخدام أربعة إصدارات من خمسة برامج مفتوحة المصدر، كما نقدم مقارنة مع ستة مقاييس استقرار موجودة مسبقاً للمقارنة .ومن أجل التحقق التجريبي، نقدم أربعة عمليات تحليل: (1) التحليل الأول يستكشف العلاقة بين مقاييس استقرارالحزمة وجهود الصيانة؛ (2) التحليل الثاني يستكشف الارتباطات بين ستة مقاييس الاستقرارالحالية والمقاييس المقترحة لاستقرارالحزمة؛ (3) التحليل الثالث يطبق المكون الرئيسي لتقديم أدلة على أن المقاييس الجديدة تلتقط بعدا جديدا في استقرارالحزمة؛ و (4) تحليل يطبق تحليلا للانحدار الخطي للتنبؤ بجهد الصيانة. وتظهرالنتائج أن نجاح مقاييس PSM يوفر مؤشرا أفضل لاستقرار الحزمة من مقاييس الاستقرار الحالية ويرتبط سلبا بجهدالصيانة. ويثبت التحليل أيضا أن مجموعة مقاييس PSM تغطي بعدا جديدا لاستقرار الحزمة وتزيد من دقة التنبؤ بجهود الصيانة. كما نقدم أداء خوارزميات الانحدار المختلفة وخوارزميات التصنيف للتنبؤ بصيانة الحزمة.

CHAPTER 1

INTRODUCTION

1.1 Overview

The widespread use of software has placed expectations on the industry to develop techniques and tools to promote quality software that is stable and easy to maintain [3, 4]. During the development life cycle, practitioners use metrics to assess and improve software quality. A number of metrics have been proposed to measure different characteristics for software products such as functionality, reliability and maintainability. Maintainability is one important property of design as software evolve to adapt changes in user requirements and operational environments [5]. Software stability contributes towards maintainability quality attribute and is one of the measures to obtain maintainable software. Stable software tends to minimize changes, improve maintainability and as a result help reduce maintenance effort [6, 7, 8].

Software stability generally falls into three main views [2, 5]. According to

the first definition, stability is the property to resist any change in the system. It means that entity of software will be called stable if it remains same between two versions of the software. The definition used by Martin [9] and Soong [10] is similar to this concept. The second definition says that due to addition or modification if an entity of the software avoids ripple effects then it is stable. So object-oriented entities (e.g. classes and packages) which does not cause ripple effects in result of the modifications will be classified as stable. This definition is used by Yau and Collofello [11], Elish and Rine [12] and Fayads [13] [14] [8]. According to the third definition, the entity has maximum stability if existing contents remain same. An entity remains stable if the addition of new contents does not affect existing contents. The definition used by Grosser et al. [15] resembles this view. In this thesis, we adopt the third definition which allows the addition of new contents while keeping the existing ones intact [5]. This implies that making any changes in the existing content may lead to an unstable artifact, while adding new features will not affect stability.

1.2 Research Problem

Researchers have developed a number of metrics to evaluate the stability quality of object-oriented applications from system, architecture and class viewpoints. System and architecture view based metrics measure stability for an application as one unit. System level stability metrics take into account changes in the number of classes and line of code, without paying attention to internal relationships and

external connections ([16, 17, 18]). On the other hand, architecture level stability metrics do take into account the change in calls between classes [19, 20, 21], however, they measure stability as a number for a whole system. However, it has been argued that internal organization of a class needs to be considered to have informative assessment of system stability [17]. Class level stability metrics focus on measuring stability of individual classes across versions and take into account line of code [16], percentage of changed and added methods [5], number of methods [22] and different class properties [5]. [23, 22, 17, 24].

In object-oriented paradigm, a package is used as the unit of organization to group relevant classes that are related through similar functionality [25, 26]. In order to facilitate maintenance, each package should be stable in a well designed object-oriented system. Package stability indicates the extend to which an individual package can tolerate evolutionary changes. This implies that any changes in the existing package may lead to an instability while adding new classes will not affect stability [5]. It is believed that a package with higher stability tend to require lower maintenance effort than the one with lower stability. In this context, if we can measure package stability, it will help designers to identify the packages with potential maintainability concerns and later take remedy measures to enhance their quality.

1.3 Motivation

There exist very few studies on software metrics to determine stability at the

package level and those studies do not cover all aspects of package stability. For example, Martin [27] presented package level instability metric as a ratio of efferent coupling to total coupling for package of a single version and does not compare two version of the software to compute package stability. Li et al. [16] calculate package level stability without considering calls between packages and classes. Some key features of object-oriented systems, such as changes in package content, intra-package and inter-package connections are not considered in package stability metrics proposed to date. Existing package stability metrics have not been validated in terms of their mathematical properties. Furthermore, research in the area of package stability metrics need empirical studies that correlation among proposed metrics and explore the relationship between package stability and software quality attributes.

1.4 Research Statement

Purpose of this research is to study package stability metrics. First, we will study existing metrics and their gaps. Then we will propose stability metrics to cover gaps of existing metrics. After proposing metrics, we will validate metrics theoretically and empirically. After validating metrics, I will perform maintenance prediction analysis using package stability metrics. Below are some research questions of this research:

- **RQ1** What are the existing package stability metric?
- **RQ2** What are the gaps in existing stability metric?

- **RQ3** What is the relationship between proposed metrics and package maintenance?
- **RQ4** What is the relationship between proposed metrics and existing stability metrics?
- **RQ5** Does proposed metrics capture new dimension by measuring missing properties?
- **RQ6** Can we use package stability metrics to predict package maintenance?
- **RQ7** Which regression technique perform better in prediction of package maintenance?
- **RQ8** Which classification technique perform better in prediction of package maintenance?

1.5 Research Methodology

In order to answer our research question we have divided our research in phases. Below are the phases of our research:

- **Phase 1: Literature Review** In this phase, we will conduct a literature review to extract stability metric that exists in the literature. This will help to identify gaps.
- **Phase 2: Stability Metric Definition** After identifying gaps, we will propose new package stability metrics to fill the gaps. In this, we will iden-

tify properties that should be measured in new metric and mathematical formulas for calculations.

- **Phase 3: Theoretical Validation** In this phase, we will theoretically validate proposed metrics against some mathematical properties and theoretical properties using theoretical framework.

- **Phase 4: Metric Tool Development** In this phase, we will develop new metrics tool from scratch to measure new proposed metrics and existing stability metrics.

- **Phase 5: Empirical Validation**: In this phase, we will collect famous open source projects. Then we will extract data by measure their stability metric values. Then we will use three set of experiments to validate proposed metrics. In first experiments, we will perform correlation analysis. In the second experiment, we will perform PCA analysis to identify that proposed metrics are capturing new dimension or not. In the third experiment we will study that can we use stability metric for maintenance prediction.

- **Phase 6: Maintenance Prediction using Regression** In this phase, we will study different regression techniques and their performance in prediction of maintenance.

- **Phase 7: Maintenancen Regression using Classification** In this phase, we will study different classification techniques and their performance in prediction of maintenance.

1.6 Thesis Contribution

In this thesis some of our contributions are given below:

- We present a new Package Stability Metrics (PSM) that accounts for changes in internal package contents, intra-package connections and inter-package connections. We use the notion of package content stability to measure the changes in the content of package classes and interfaces. Intra-package contents stability measures the change in internal connections of a package. Inter-package stability measures the change in a package's external connections.

- We develop custom metric tool for calculation of stability metrics values of open source projects.

- We study the validity of PSM both theoretically and empirically. The theoretical validation involves analyzing the compliance of PSM with the properties proposed by Kitchenham et al. [1].The empirical validation involves analyzing the correlation between PSM and maintenance effort.

- We also present a comparison with three existing package stability metrics. Furthermore, we also develop maintenance effort prediction model to gather empirical evidence that proposed metrics better relates to package stability quality than other comparable metrics.

The data for the empirical validation is collected from five open source software involving desktop budgeting application, graphical tool, online game, data

integration tool and a scheduling system. For each open source software, we have used four different versions with at least one year difference between their respective release date. The results show that PSM provides better indication of package stability than the existing alternatives as it considers package's functional, internal behavioral and structural properties. The PSM points out the unstable packages that may need to be refactored.

In particular, these results could improve the understanding of the value of client usage context in package cohesion, guide the development of better fault-proneness prediction models in practice, and also help developers to identify the packages with higher defect density.

1.7 Thesis Outline

The remainder of this thesis is organized as follows. Section 2 summarizes the related work. In Section 3, we define the PSM metrics, and in Section 4, we validate PSM metrics theoretically and empirically. For empirical validation we have performed four analysis: (1) first analysis explores the correlation between package stability metrics and maintenance effort; (2) second analysis explores the correlations among six existing stability metrics and proposed package stability metrics; (3) third analysis applies principal component analysis to provide evidence that the new metrics captures new dimension of package stability; and (4) forth analysis applies linear regression analysis for maintenance effort prediction.. In section 6 we discusses the threats to the validity of our study. In section 5 we

compare the performance of regression and classification algorithm, by predicting maintenance effort using six existing metric, PSM metrics suite, two package cohesion metrics, and two package coupling metrics. Section 6 concludes the thesis and outlines the directions for future works.

CHAPTER 2

RELATED WORK

In this section, we review existing stability metrics for object-oriented systems at system, architecture, class and package levels.

2.1 System Stability

Soong [10] used program information to quantify program stability and reliability. According to them, stability is the property of program with good information structure which helps it to resist changes. They quantify information structure of programs in order to measure their stability using techniques like connectivity matrix and random Markovian process. According to Yau and Collofello [11] stability is the resistance to potential ripple effects due to changes in program. They presented metric for program stability, which calculates logical ripple effect of changes to a program. They also provided an algorithm for calculation of program stability and also for normalization. Garland et al. [28] identified factors which reduce the stability of the software and make it unstable. They also

proposed techniques to make the system more stable.

Li et al. [16] shared that stability tells us that how mature the implementation and design of a software is and it can be used as an indication of project progress. They proposed System Implementation Instability (SII) metric, which calculates the percentage of change in LOC of the whole system between two version in order to calculate changes in the implementation of the object-oriented system. They also presented System Design Instability (SDI) metric, which calculates the percentage of added classes, deleted classes, and classes with changed names. SDI measures the changes in the design of the object-oriented system. They empirically validated their metrics by conducting correlation analysis with C&K [29] metrics.

Alshayeb and Li [17] updated SDI metric by adding a new factor, change in the inheritance hierarchy of classes, in calculations. They validated SDI by conducting an empirical study on two object-oriented system developed by using agile methods (XP). They concluded that SDI has a correlation with XP activities and it can be used for estimation and re-planing of software project developed by agile methods. Olague et al. [24] also improved SDI metric and proposed entropy-based SDI metric (SDIe) to remove the spikes in measurements and make calculations easy. They replaced "percentage of classes with changed names" factor with new two factors; percentage of changed classes and unchanged classes. Because the change in name of classes is not easy to track. Change in class means, change in the class features for perfection in its design and unchanged class means, perfectly designed class that contributes to the stability of the system. They

validated new metric using maintenance data of commercial software which was developed by agile methods.

Raemaekers et al. [30] studied the stability of third-party libraries. They proposed new metric suite to cover four different aspects of third-party libraries. To assess interface stability they measured the number of removed methods with weights (WRM). To calculate volatility of library they measure the amount of change in existing methods (CEM). For determining that library is in maintenance phase or in active development phase, they took the ratio between the amount of change in new methods and amount of change in old methods (RCNO). To calculate expansion rate of the library they measure the percentage of new methods (PNM). Abu Hassan and Alshayeb [18] studied stability at the model level. They proposed new stability metrics for three different UML diagrams view; structural view, behavioral view, and functional view. In order to cover structural view, they studied ten properties of class diagram and identified eight properties that actually impact class diagram stability. Those eight properties cover change in classifier (class) type and relationships. To cover functional view, they studied eight properties of use case diagram and constructed its stability definition based on the change in use case type, use case relationships and actor relationships. For the behavioral view, they investigated sequence diagram by identifying nine properties that can affect its stability. Their sequence diagram stability definition is based on changes in message receiver, message caller, message types and message order. Table 2.1 provides the summary of system level stability metrics.

Table 2.1: System Stability Literature Review

Reference	Level	Artifact	Validation	Stability Metric Detail
Soong [10]	System	Code	-	Calculates system stability using information structure of program.
Yau and Collofello [11]	System	Code	Case Study and Theoretically	System stability is calculated from changes in logical ripple effect of programs.
Li et al. [16]	System	Code	Theoretically and Experiments	Measures percentage of change in LOC between two version in order to calculate System Implementation Instability (SII) metric.
Li et al. [16]	System	Code	Theoretically and Experiments	Measures change (addition, deletion and updation) in name to classes to calculate system level instability (SDI).
Alshayeb and Li [17]	System	Code	Theoretically and Experiments	Updating previous study of Li et al. [16] by including change in inheritance factor in calculation of system instability.
Olague et al. [24]	System	Code	Theoretically and Experiments	This study improved SDI metric [16] and proposed entropy-based metric that removes spikes.
Raemaekers et al. [30]	System	Code	Experiments	Proposed stability metrics for third-party libraries.
Abu Hassan and Alshayeb [18]	Model	UML	Theoretically and Case Study	Proposed new stability metrics for class diagram, use case diagram, and sequence diagram.

2.2 Class Stability

Li et al. [16] presented three metrics for instability; System Implementation Instability (SII), System Design Instability (SDI) and Class Implementation Instability (CII). SII and SDI are system level metrics and are discussed earlier. Class Implementation Instability (CII) calculates instability of classes by measuring the percentage of line of code that are changed between two version of software. Grosser et al. [15] studied the class stability and presented new metric using Case-Based

Reasoning (CBR) which calculates structural similarities between classes. They also evaluated their metric performance for finding quality challenges using other metrics for inheritance, complexity, coupling and cohesion. They validated their metric prediction quality by comparing its results with a classical learning method TDIDT, using 10-fold cross-validation and leave-one-out validation. Grosser et al. [23] improved their metric by adding new stress factor in calculations. Due to change in requirements, the responsibilities of some classes increased in the new version of software. This is called stress factor on classes and authors included it in improved class stability metric. They calculated stress factor by measuring four modifications; new class methods, change in class ancestors, change in class descendants and change in classes on which a class is dependent.

Rapu et al. [22] presented class level stability metrics by using a number of methods. According to him if the number of functions of a class between different versions remains same then the class has maximum stability. They also proposed a technique to detect flawed classes like god classes and data classes. For validation, they applied their technique and metric on three case studies. Alshayeb et al. [5] proposed new Class Stability Metric (CSM). They identified eight properties of classes to calculate class stability. Those properties include class properties (class name, access level, class interface name and inherited class name), class variable properties (class variable name and access type), and class method properties (method signature and body). Change in these properties between versions will reduce the stability of class but newly added properties will

have no impact. If there is no change in these properties then class is stable. Alshayeb et al. [31] conducted study to compare performance of artificial neural network and support vector machine for prediction of class stability using different software design measurements. They shared that the proposed prediction models give good prediction for class stability. Table 2.2 provides the summary of system level stability metrics.

Table 2.2: Class Stability Literature Review

Reference	Level	Artifact	Validation	Stability Metric Detail
Li et al. [16]	Class	Code	Theoretically and Experiments	Measures class implementation instability (CII) by measuring the percentage of change in line of code.
Grosser et al. [15] [23]	Class	Code	Experiments	Presented new class stability metric based on case-based reasoning using other metrics for inheritance, complexity, coupling and cohesion.
Rapu et al. [22]	Class	Code	Case Study	Counts change in number of methods of a class to calculate its stability.
Alshayeb et al. [5]	Class	Code	Theoretically and Experiments	Proposed class stability metric (CSM), which calculates changes in eight properties of classes to measure its stability.

2.3 Architecture Stability

Ahmed et al. [19] used similarity metrics to calculate the architectural stability of an object-oriented system. For similarity calculation, they compared the architectures of different version to the base version. A low value shows that architecture is not stable whereas high value identifies stable architecture. Sethi et al. [32] proposed measurements for software architecture modularity and stability

using aspect-oriented concept. According to their metric environment conditions can affect the architecture stability. They shared that good modularity can increase the stability of architecture. Bansiya [33] proposed new methodology to assess the stability of architecture using object oriented structural characteristics. Those structural characteristics include the number of classes, different class hierarchy metrics, the number of parents, the number of functions and coupling of class. They shared that most of the changes in structural characteristics are due to newly added classes, reassignment of responsibilities to classes and change in collaboration between classes.

Aversano et al. [20] proposed two architectural stability metrics Core Design Instability (CDI) and Core Call Instability (CCI). CDI calculates changes in a number of packages and CCI calculates the change in calls between packages. In their metrics, smaller values mean fewer modifications and the architecture is stable. They did not normalize their metric, so for comparison threshold values are required. Ebad and Ahmed [21] proposed new metrics for Architecture Stability Metric (ASM) of object-oriented software by calculating inter-package calls instead of inter-package relationships. According to them change in cross-component (inter-package) calls is costly and must be reduced to improve stability. They also validated ASM metric using two open source software by comparing ASM with lines of code changes. Alenezi [34] studied the factors that affect software architecture stability and understandability by doing detailed literature review. He also discussed that why theoretical and empirical validations are important

and how researchers have done it. Handani and Rochimah [35] investigated the relationship between features volatility and architecture stability. They calculated architecture stability by using Aversano et al. [20] and Constantinou and Stamelos [36] metric definition which calculate changes in external and internal elements of architecture. They measures features volatility by counting changes in features. To find the relationship they conducted Pearson correlation analysis. Alenezi and Khellah [37] proposed a simple technique to measure architecture stability by aggregating the package level stability using Martin's package instability metric. Table 2.3 provides the summary of system level stability metrics.

2.4 Need for a New Package Stability Metrics

A number of software stability metrics have been presented in literature at system, architecture and class levels, but at the package level, very few studies exist, as shown in Tables 2.2, 2.1 and 2.3. There is only one package level stability metrics [27] presented in literature. Martin [27] presents package level instability metric based on efferent coupling properties of a single version of software. Martin's package instability metric does not compare two version of the software to compute package stability. Architecture stability metrics can be used to measure package stability metric. But existing architecture stability metrics does not cover all aspect of package stability. Existing class and system level stability metric cannot be used to measure package stability.

Some researchers have used interactions between classes to calculate stability

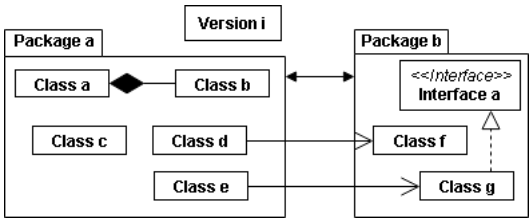


Figure 2.1: Example version i

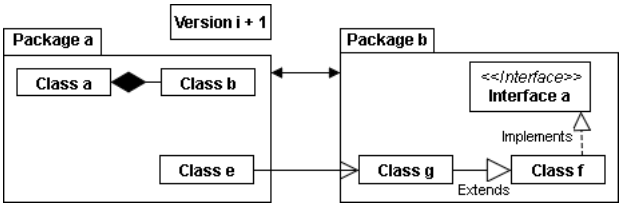


Figure 2.2: Example version i + 1

at class and architecture level. For example, Alshayeb et al [5] calculates stability of a class in terms of change in content (e.g class name, class variable, method signature etc.) and it's inheritance relationship with other classes. This metrics does not cover other possible interactions between classes such as association, aggregation and dependency relationships. Similarly, Ebad and Ahmed [21] presented architecture stability metrics in terms of inter packages method calls. However, method call do not represent other possible interactions between classes such as inheritance and association relationships; and they do not consider intra-package interactions.

Hence, the existing architecture level stability metrics can not be used to measure package level stability as do they not holistically consider changes in contents of packages, changes in intra-package interactions and changes in inter-package interactions. For example, we have a software with two packages, namely package A and package B. Figure 2.1 shows version i of the system while Figure

2.2 shows version i+1 of the system. The content and structure of package A have changed due to deletion of class C; and merger of class D and E. On the other hand, contents of package B have not changed but package B's structure has changed due to introduction of inheritance relationship between classes G and F. The existing stability metrics do not the measure the changes in both content and structure of packages. Measuring package stability only based on package's content will not be able to identify the structural changes in package B.

Furthermore, in version i, class D in package A has an association relationship with class F in package B, however, in version i+1, only class E has an association relationship with class G; class D is merged in class E; and class G also extends class F. This indicates that behavior of both packages A and B have changed due the modifications in calling order between classes of same package as well as across packages. These changes will result in behavioral instability of both packages A and B. So in order to measure behavioral stability, we have to consider changes in relationships between entities of same package as well as across packages. The existing stability metrics fail to identify behavioral changes both in packages and across packages.

Hence, there is a need to include both intra-package and inter-package method calls to measure behavioral stability of a package. For inter-package method calls, we should not consider the direct connection between packages and ignore method calls between entities (i.e. classes, interfaces) of different packages. As shown in Figure 2.1 and Figure 2.2, if we only calculate direct package connection between

package A and B, then both packages are stability. However, the number of method calls between entities of package A and B have changed (version i has two method calls while version i+1 has one method call).

In a nutshell, the package metrics proposed to date have one or more of the following limitations: (1) they do not consider functional, structural and behavioral aspects of an object oriented system; (2) they lack theoretical validation; and (3) they have not been empirically validated in terms of their relationship with quality attributes such as maintenance effort.

Table 2.3: Architecture Stability Literature Review

Reference	Level	Artifact	Validation	Stability Metric Detail
Ahmed et al. [19]	Architecture	UML	Case Study	Used similarity metrics to calculate stability of object-oriented system's architecure.
Bansiya [33]	Architecture	UML	Case Study	Proposed new methodology to assess the stability of architecture using 9 object oriented metrics.
Aversano et al. [20]	Architecture	UML	-	Measure architecure instability using the change in number of packages and calls between packages.
Ebad and Ahmed [21]	Architecture	UML	Theoretically and Case Study	Proposed architecture stability metric (ASM) which uses change in inter-packages calls.
Alenezi and Khellah [37]	Architecture	Code	-	Proposed a technique to measure architecture stability by aggregating the package level stability using Martin's package instability metric.
Martin [27]	Package	Code	-	Calculates packages instability by taking ratio of efferent coupling (Ce) and total coupling (Ce+Ca).
Proposed Metrics	Package/ Architecture	Code	Theoretically and Empirically	Calculates packages content stability (PCS) by measuring unchanged count of eight different package content properties.
Proposed Metrics	Package/ Architecture	Code	Theoretically and Empirically	Calculates inter-packages interactions stability (EPIS) by measuring unchanged count of ten different inter-package interactions.
Proposed Metrics	Package/ Architecture	Code	Theoretically and Empirically	Calculates internal package interactions stability (IPIS) by measuring unchanged count of ten different interactions.

CHAPTER 3

PACKAGE STABILITY METRIC DEFINITION

In this chapter we presents definition of our proposed package stability metrics. The package stability metrics introduced in this thesis considers changes in package contents, intra-package interactions and inter-package interactions. First, we identify the properties that affect package stability and present new package stability metrics that uses the identified properties to measure package stability.

3.1 Package Stability Properties

In order to identify properties for three package stability aspects; content, internal interaction and external interaction, we analyzed UML metamodel. Benefit of UML metamodel is that it is independent of programing language syntax and it cover all possible properties of a standard object oriented softwares.

Package Content Stability Properties: The package content stability prop-

ertiess model the changes in contents of classes and interfaces. We consider eight properties namely, class access level, class name, class variables declaration, class functions declaration, class body, interface access level, interface name and interface functions to measure package content stability for the version with respect to the base version. A property is considered unchanged if it has not been changed between the base version i and version i+1. Table 3.1 shows details about the eight different properties that affect package content stability.

Table 3.1: Package Content Stability Factors

	Properties	Description
1	Class Access Level	Class access level can be public to private and it restrict the access scope a class in software.
2	Class Name	If class is used by many other entities in software then changing its name will effect those entities too.
3	Class Variables Declaration	In this property, we will calculate change in class variable name, access level and data type.
4	Class Functions Declaration	In this property, we will measure modification in class function name, access level, return type, parameters name, parameters data type and number of parameters.
5	Class Body	In this property, we will check the change in line of code of whole class.
6	Interface Access Level	Interface access level can be public to private and it restrict the access scope an interface in software.
7	Interface Name	Change in interface name effect other entities too which depends on it.
8	Interface Functions	In this property, we will measure modification in interface function name, access level, return type, parameters name, parameters data type and number of parameters.

Intra-Package Stability (IPIS) Properties: The intra-package stability properties model the changes in direct interactions between classes and interfaces of a single package. The intra-package interactions are classified into three

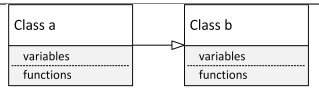
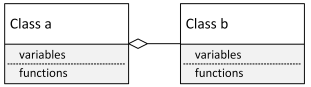
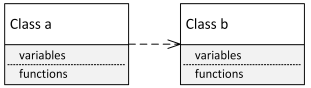
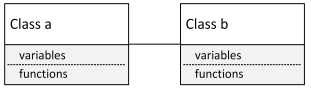
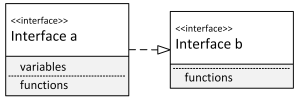
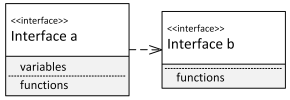
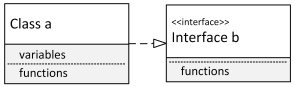
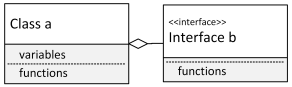
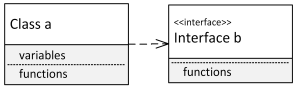
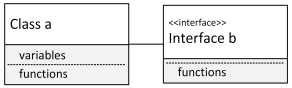
main categories, namely, interactions between classes, interaction between interfaces and interaction between classes and interfaces. The interactions between classes are modeled using inheritance, aggregation, composition, dependency and association relationships. The interactions between interfaces are modeled using inheritance and dependency relationship. Similarly, the interactions between classes and interfaces are modeled using inheritance, aggregation, composition, dependency and association relationship.

Inter-Package Stability (EPIS) Properties: The inter-package stability properties model the changes in direct interactions between packages at classes and interfaces level of respective packages in system. The inter-package interactions are classified into three main categories, namely, interactions between classes, interaction between interfaces and interaction between classes and interfaces across different packages. The interactions between classes of different packages are modeled using inheritance, aggregation, composition, dependency and association relationships. The interactions between interfaces of different packages are modeled using inheritance and dependency relationship. Similarly, the interactions between classes and interfaces of different packages are modeled using inheritance, aggregation, composition, dependency and association relationship.

Table 3.2 shows details about the different interactions properties that affect both intra-package and inter-package stability.

Type of Changes: There are four types of changes [21] that can occur on package properties when two versions of the same package are compared. The

Table 3.2: Package internal/external Connections Stability Factors

	Factor	Relationship	Description	Figure
1	Between classes	Inheritance	If class a inherit class b then class a has inheritance relationship with class b.	
2		Aggregation and Composition	If class a has class variable type of class b then class b have aggregation relationship with class a. We have combined both composition and aggregation relationship in this property.	
3		Dependency	If function of class a have class b as one of the parameters then class a has dependency relationship with class b.	
4		Association	If class a create instance of class b in function body then both classes have association relationship.	
5	Between interfaces interaction	Inheritance	If interface a implement interface b then interface a has inheritance relationship with interface b.	
6		Dependency	If function deceleration of interface a have interface b as one of the parameters then interface a has dependency relationship with interface b.	
7	Between classes and interfaces	Inheritance	If class a implement interface b then class a has inheritance relationship with interface b.	
8		Aggregation and Composition	If class a has class variable type of interface b then interface b have aggregation relationship with class a.	
9		Dependency	If function signature of class a have interface b as one of the parameters then class a has dependency relationship with interface b and vice versa.	
10		Association	If class a create instance of type of interface b in function body then both have association relationship.	

different types of changes are as follows:

- Addition: An entity that does not exist in version i and added in version i + 1 then it will fall in addition property.
- Deletion: An entity that exist in version i and removed from version i + 1.
- Modification: An entity that exist in version i and modified in version i + 1.
- Unchanged: An entity that exist in version i and remains same in version i + 1.

We consider version i+1 of a package completely stable if none of its properties have changed. On the other hand, we consider version i+1 of a package to be completely instable if all of its properties have changed. Hence, we measure package stability by counting the unchanged properties between version i and version i+1. Table 3.3 and 3.4 summaries the considered package properties and how each property is counted.

Maximum Possible Change: To measure the package stability, we assume that each of the identified package property has the same weight. Therefore, we calculate the stability of each property and take a sum of stability values of all applicable properties to compute package stability. Furthermore, to normalize measurements of each package property, we use the concept of maximum possible change [38] that can happen to each property with respect to version i. Table

3.5 and 3.6 presents the maximum possible change for each property used to calculate package stability of an object oriented system.

Table 3.3: Package Contents Type of Change

	Metric	Action	Description	Count
1	Class Access Level	un-changed	if class access level is not deleted or modified from version i to version i + 1.	+1
2	Class Name	un-changed	if class name is not deleted or modified from version i to version i + 1.	+1
3	Class Variables Declaration	un-changed	if class name is not deleted or modified from version i to version i + 1.	+1
4	Class Functions Declaration	un-changed	if class function name, access level, return type, parameters name, parameters data type and number of parameters are not deleted or modified from version i to version i + 1.	+1
5	Class Body	un-changed	In this property, if number of line of whole class are not deleted or modified from version i to version i + 1.	+1
6	Interface Access Level	un-changed	if interface access level is not deleted or modified from version i to version i + 1.	+1
7	Interface Name	un-changed	if interface name is not deleted or modified from version i to version i + 1.	+1
8	Interface Functions	un-changed	if interface function name, access level, return type, parameters name, parameters data type and number of parameters are not deleted or modified from version i to version i + 1.	+1

3.2 The Package Stability Metrics Definition

The package properties identified in Section 3.1 are used to calculate package stability that compares the target package (version i+1) with its previous version

Table 3.4: Package internal/external Connections Type of Change

	Metric	Relationship	Action	Description	Count
1	Between classes	Inheritance	unchanged	if class a inherits class b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
2		Aggregation and Composition	unchanged	if class a has a class variable of type class b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
3		Dependency	unchanged	if the function of class a has a parameter of type class b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
4		Association	unchanged	if the function of class a creates an instance of class b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
5	Between interfaces	Inheritance	unchanged	if interface a inherits interface b in version i and this connection remains unchanged in version i + 1 then the stability of interface a's package will be increased.	+1
6		Dependency	unchanged	if the function decleration of interface a has a parameter of type interface b in version i and this connection remains unchanged in version i + 1 then the stability of interface a's package will be increased.	+1
7	Between classes and interfaces	Inheritance	unchanged	if class a implements interface b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
8		Aggregation and Composition	unchanged	if class a has a class variable of type interface b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1
9		Dependency	unchanged	if the function of class a has a parameter of type interface b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased and vise versa.	+1
10		Association	unchanged	if the function of class a creates an instance of interface b in version i and this connection remains unchanged in version i + 1 then the stability of class a's package will be increased.	+1

Table 3.5: Package Content Maximum Possible Change value

	Metric	Maximum Possible value	When it occur
1	Class Access Level	total class count of the package in version i	if the access levels of all classes of the package are modified in version i + 1
2	Class Name	total class count of the package in version i	if the names of all classes of the package are modified in version i + 1
3	Class Variables Declaration	sum of all classes' variables count of the package in version i	if any property from access levels, names or data type of all classes' variables of the package are modified in version i + 1
4	Class Functions Declaration	sum of all classes' functions count of the package in version i	if any property from names, access levels, return types, parameters names, number of parameters or parameters data type of all classes' functions of the package are modified in version i + 1
5	Class Body	total class count of the package in version i	if number of line of all class of the package are modified in version i + 1
6	Interface Access Level	total interface count of the package in version i	if the access levels of all interfaces of the package are modified in version i + 1
7	Interface Name	total interface count of the package in version i	if the names of all interfaces of the package are modified in version i + 1
8	Interface Functions	sum of all interfaces' functions count of the package in version i	if any property from names, access levels, return types, parameters names, number of parameters or parameters data type of all interfaces' functions of the package are modified in version i + 1

Table 3.6: Package internal/external Connections Maximum Possible Change value

	Metric	Relationship	Maximum Possible Change value	When it occur
1	Between classes	Inheritance	total number of classes of a package that has inheritance relationship with other classes in version i.	if all classes' inherit class names of a package from version i are modified in version i + 1
2		Aggregation and Composition	total number of classes of a package that has aggregation relationship with other classes in version i.	if all such class variables of all classes of a package which have data type as of other classes from version i are modified in version i + 1
3		Dependency	total number of classes of a package that has dependency relationship with other classes in version i.	if all such function parameters of all classes of a package which have data type as of other classes from version i are modified in version i + 1
4		Association	total number of classes of a package that has association relationship with other classes in version i.	if all such variable in functions of all classes of a package which have data type as of other classes from version i are modified in version i + 1
5	Between interfaces	Inheritance	total number of interfaces of a package that has inheritance relationship with other interfaces in version i.	if all interfaces' inherit interfaces names of a package from version i are modified in version i + 1
6		Dependency	total number of interfaces of a package that has dependency relationship with other interfaces in version i.	if all such function parameters of all interfaces of a package which have data type as of other interfaces from version i are modified in version i + 1
7	Between classes and interfaces	Inheritance	total number of classes of a package that has implement relationship with other interfaces in version i.	if all classes' interface names of a package from version i are modified in version i + 1
8		Aggregation and Composition	total number of classes of a package that has aggregation relationship with other interfaces in version i.	if all such class variables of all classes of a package which have data type as of other interfaces from version i are modified in version i + 1
9		Dependency	total number of classes of a package that has dependency relationship with other interfaces and vice versa in version i.	if all such function parameters of all classes of a package which have data type as of other interfaces from version i are modified in version i + 1 and vice versa
10		Association	total number of classes of a package that has association relationship with other interfaces in version i.	if all such variable in functions of all classes of a package which have data type as of other interfaces from version i are modified in version i + 1

(version i). First step in defining package stability metric is to calculate the stability value for each package property.

- **Package Property Stability:** The package property stability is ratio of number of unchanged properties divided by maximum possible changes of that property. The package property stability is formally defined as follows:

$$St_{\text{Property}} = \frac{NOU_{\text{Property}}}{MPC_{\text{Property}}} \tag{3.1}$$

where ($\text{NOU}_{\text{Property}}$) is the number of unchanged items of the property and ($\text{MPC}_{\text{Property}}$) is the maximum possible changes for the property. Table 3.3 and 3.4 shows the details for calculation of a number of unchanged counts for different package properties. Table 3.5 and 3.6 presents details for maximum possible changes for different package properties.

Next, the package stability property values are used to calculate package content stability, intra-package interaction stability and inter-package interaction stability as follows:

Package Content Stability (PCS): Package content stability is the average of all package content properties. Formally, the package content stability of a package is defined as follows:

$$PCS = \frac{Sumof8Packagecontentproperties}{PropertiesCount} \tag{3.2}$$

Sum of Eight Package Content properties = $\text{St}_{\text{classAL}} + \text{St}_{\text{className}} + \text{St}_{\text{classVar}}$

$+ \mathrm{St}_{\mathrm{classFunc}} + \mathrm{St}_{\mathrm{classBody}} + \mathrm{St}_{\mathrm{intAL}} + \mathrm{St}_{\mathrm{intName}} + \mathrm{St}_{\mathrm{intFunc}}$

where PCS is package content stability; $\mathrm{St}_{\mathrm{classAL}}$ is the stability of package's class access level property ; $\mathrm{St}_{\mathrm{className}}$ is the stability of package's class name property ; $\mathrm{St}_{\mathrm{classVar}}$ is the stability of package's class variables declaration property ; $\mathrm{St}_{\mathrm{classFunc}}$ is the stability of package's class functions declaration property ; $\mathrm{St}_{\mathrm{classBody}}$ is the stability of package's class body property ; $\mathrm{St}_{\mathrm{intAL}}$ is the stability of package's interface access level property ; $\mathrm{St}_{\mathrm{intName}}$ is the stability of package's interface name property and $\mathrm{St}_{\mathrm{intFunc}}$ is the stability of package's interface functions property. Properties Count is the total count of properly defined properties.

Table 3.1 presents details about the eight different properties that affect package content stability. Furthermore, it is important to note that if a property does not exit in version i of a package, then it is excluded from the package content stability measurements.

Intra-package Interaction Stability (IPIS): Intra-package interaction stability is the average of all intra-package stability properties. Formally, the intra-package stability is defined as follows:

$$IPIS = \frac{SumofTenIntraPackageInteractionproperties}{PropertiesCount} \tag{3.3}$$

Sum of Ten Intra Package Interaction properties $= \mathrm{St}_{\mathrm{IbcInh}} + \mathrm{St}_{\mathrm{IbcAgg}} + \mathrm{St}_{\mathrm{IbcDep}} + \mathrm{St}_{\mathrm{IbcAss}} + \mathrm{St}_{\mathrm{IbiInh}} + \mathrm{St}_{\mathrm{IbiDep}} + \mathrm{St}_{\mathrm{IbciInh}} + \mathrm{St}_{\mathrm{IbciAgg}} + \mathrm{St}_{\mathrm{IbciDep}} +$

$St_{IbciAss}$

where IPIS is intra-package connections stability; St_{IbcInh} is internal package between classes inheritance connections stability; St_{IbcAgg} is internal package between classes aggregation and composition connections stability; St_{IbcDep} is internal package between classes inheritance connections stability; St_{IbcAss} is internal package between classes association connections stability; St_{IbiInh} is internal package between interfaces inheritance connections stability; St_{IbiDep} is internal package between interfaces dependency connection stability; $St_{IbciInh}$ is internal package between class and interface inheritance connections stability; $St_{IbciAgg}$ is internal package between class and interface aggregation and composition connection stability; $St_{IbciDep}$ is internal package between class and interface dependency connections stability and $St_{IbciAss}$ is internal package between class and interface association connections stability. Properties Count is the total count of properly defined properties.

Table 3.2 presents details about the ten different properties that affect intra-package stability. Furthermore, it is important to note that if a property does not exit in version i of a package, then it is excluded from the intra-package interaction stability measurements.

Inter-package Interaction Stability (EPIS): Inter-package interaction stability is the average of all inter-package stability properties. Formally,

the inter-package stability is defined as follows:

$$EPIS = \frac{SumofTenInterPackageInteractionsproperties}{PropertiesCount} \tag{3.4}$$

Sum of Ten Inter Package Interactions properties = St_{EbcInh} + St_{EbcAgg} + St_{EbcDep} + St_{EbcAss} + St_{EbiInh} + St_{EbiDep} + $St_{EbciInh}$ + $St_{EbciAgg}$ + $St_{EbciDep}$ + $St_{EbciAss}$

where EPIS is inter-package connections stability; St_{EbcInh} is external package between classes inheritance connections stability; St_{EbcAgg} is external package between classes aggregation and composition connections stability; St_{EbcDep} is external package between classes inheritance connections stability; St_{EbcAss} is external package between classes association connections stability; St_{EbiInh} is external package between interfaces inheritance connections stability; St_{EbiDep} is external package between interfaces dependency connection stability; $St_{EbciInh}$ is external package between class and interface inheritance connections stability; $St_{EbciAgg}$ is external package between class and interface aggregation and composition connection stability; $St_{EbciDep}$ is external package between class and interface dependency connections stability; and $St_{EbciAss}$ is external package between class and interface association connections stability. Properties Count is the total count of properly defined properties.

Table 3.3 resents details about the ten different properties that affect inter-package stability. Furthermore, it is important to note that if a property

does not exit in version i of a package, then it is excluded from the inter-package interaction stability measurements.

- **Package Stability Metrics (PSM):** Finally, package stability metrics is the average of package content stability, intra-pacakge stability and inter-package stability. Formally, the package stability metrics is defined as follows:

$$PSM = Stability_{\text{Package}} = \frac{PCS + IPIS + EPIS}{AspectCount} \tag{3.5}$$

where 'Aspect Count' is the total count of aspects which are applicable for package.

3.3 Package Stability Metrics Measurement for Point of Sales System - An Example

In this section, we have selected one example of point of sales system's class diagram with two version i and i + 1 for calculations of PCS, IPIS and EPIS. Figure 3.1 is the class diagram for version i class diagram and figure 3.2 is the class diagram for version i + 1 class diagram.

As first step, we will calculate the stability of properties. In table 3.8, we have calculated package content properties stability. In table 3.9, calculations of intra-package connections stability are presented and in table 3.10, measurements of inter-package connections stability are done.

After calculation of properties stability, we need to take averages to calculate

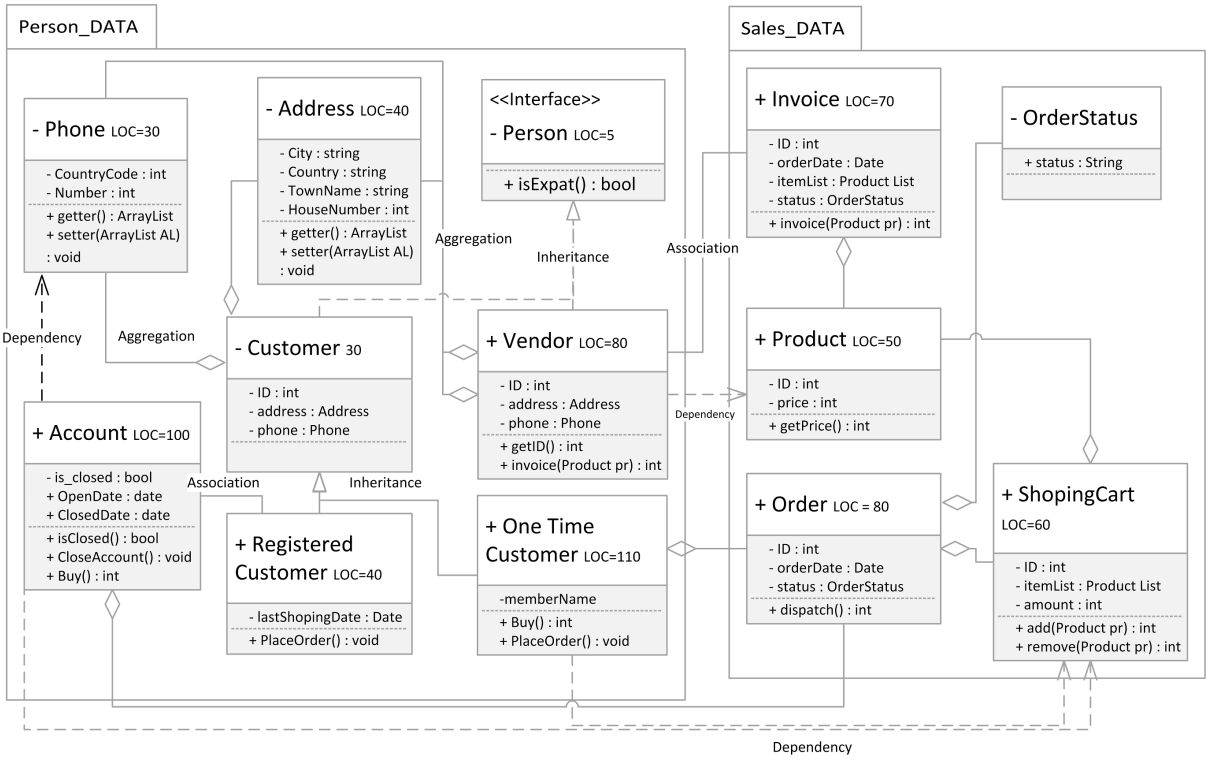


Figure 3.1: Example of package stability calculation Version i

Table 3.7: Calculation of Package Aspects and Overall Stability

	Aspect	PersonDATA	SalesDATA
1	PCS	0.298844538	0.738461538
2	IPIS	0.2	0.75
3	EPIS	0.5	not applicable
4	Overall Stability	0.332948179	0.744230769

PCS, IPIS and EPIS. Table 3.7 contains the calculation details. As package SalesData had no external connections, so EPIS is not applicable and we have considered only PCS and IPIS for overall stability calculation. So from overall package stability package PersonDATA is 33.3%stable and package SalesDATA is 74.4% stable.

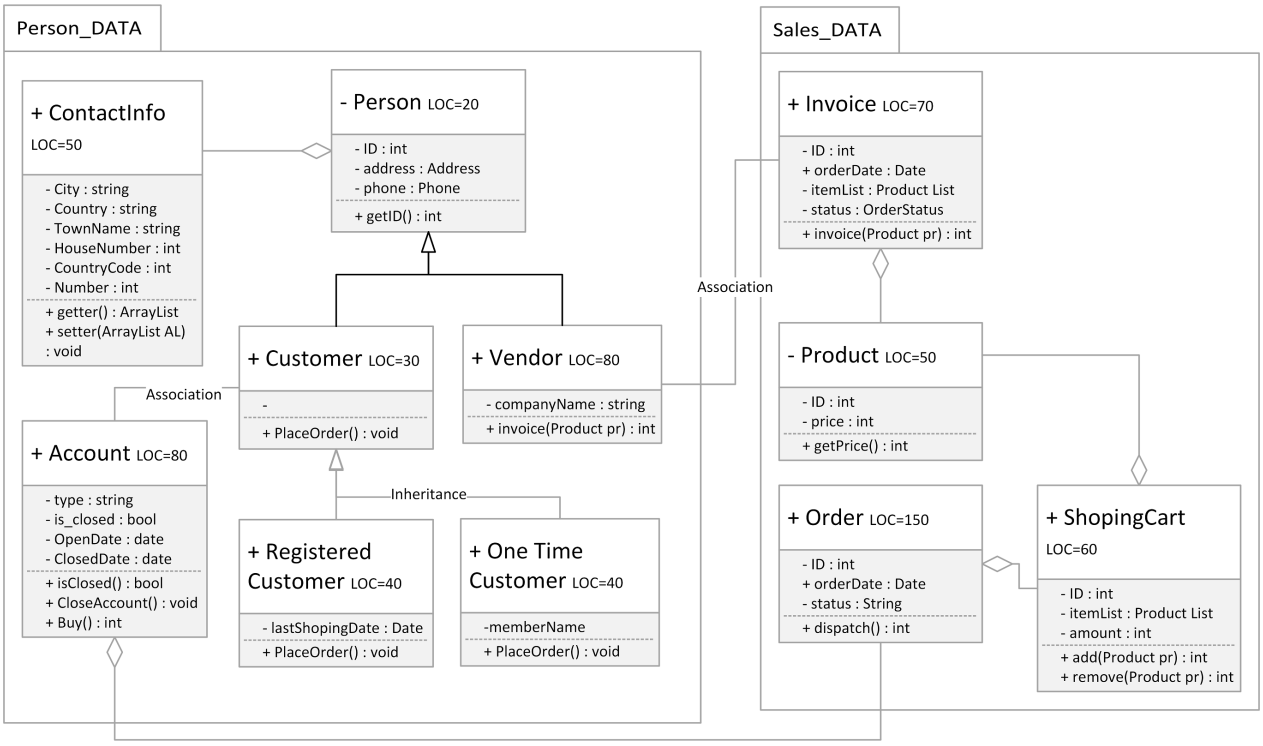


Figure 3.2: Example of package stability calculation Version i + 1

Table 3.8: Calculation of example Package Content Stability Factors

	Property	Package PersonDATA's unchanged count	Package's PersonDATA MOV	PersonDATA's Stability	Package SalesDATA's unchanged count	Package SalesDATA's MOV	PersonDATA's Stability
1	Class Access Level	4	7	0.571428571	3	5	0.6
2	Class Name	5	7	0.714285714	4	5	0.8
3	Class Variables Declaration	3	17	0.176470588	9	13	0.692307692
4	Class Functions Declaration	6	12	0.5	5	5	1
5	Class Body	3	7	0.428571429	3	5	0.6
6	Interface Access Level	0	1	0	0	0	not applicable
7	Interface Name	0	1	0	0	0	not applicable
8	Interface Functions	0	1	0	0	0	not applicable

Table 3.9: Calculation of Intra-Package Connections Stability Factors

	Property	Relationship	Package Person-DATA's unchanged count	Package's Person-DATA MOV	Person-DATA's Stability	Package Sales-DATA's unchanged count	Package Sales-DATA's MOV	Person-DATA's Stability
1	Between classes	Inheritance	2	2	1	0	0	not applicable
2		Aggregation and Composition	0	4	0	3	4	0.75
3		Dependency	0	1	0	0	0	not applicable
4		Association	0	1	0	0	0	not applicable
5	Between interfaces	Inheritance	0	0	not applicable	0	0	not applicable
6		Dependency	0	0	not applicable	0	0	not applicable
7	Between classes and interfaces	Inheritance	0	2	0	0	0	not applicable
8		Aggregation and Composition	0	0	not applicable	0	0	not applicable
9		Dependency	0	0	not applicable	0	0	not applicable
10		Association	0	0	not applicable	0	0	not applicable

Table 3.10: Calculation of Inter-Package Connections Stability Factors

	Property	Relationship	Package Person-DATA's unchanged count	Package's Person-DATA MOV	Person-DATA's Stability	Package Sales-DATA's unchanged count	Package Sales-DATA's MOV	Person-DATA's Stability
1	Between classes	Inheritance	0	0	not applicable	0	0	not applicable
2		Aggregation and Composition	1	2	0.5	0	0	not applicable
3		Dependency	0	3	0	0	0	not applicable
4		Association	1	1	1	0	0	not applicable
5	Between interfaces	Inheritance	0	0	not applicable	0	0	not applicable
6		Dependency	0	0	not applicable	0	0	not applicable
7	Between classes and interfaces	Inheritance	0	0	not applicable	0	0	not applicable
8		Aggregation and Composition	0	0	not applicable	0	0	not applicable
9		Dependency	0	0	not applicable	0	0	not applicable
10		Association	0	0	not applicable	0	0	not applicable

CHAPTER 4

THEORETICAL AND EMPIRICAL VALIDATION

In this chapter, we validate our proposed PSM metrics suite theoretically and empirical. For Theoretical validation we have used two frameworks; (1) Kitchenham et al. [1] and (2) Hassan [2]. For empirical validation, we have used correlation analysis, principal component analysis [39, 40] and linear regression [41].

4.1 Theoretical Validation

For theoretical validation, we need to evaluate a metric against some properties proposed by researchers. We have validated our metrics by using two frameworks proposed by Kitchenham et al. [1] and Hassan [2].

4.1.1 Kitchenham et al. [1] Framework

Kitchenham et al. framework has four mathematical properties which can be used to validate any software metric. Evaluation of our metrics (PCS, IPIS, EPIS and PSM) against these four properties are below.

- **Property 1: A metric must distinguish between different entities.** Let P1 and P2 be two packages with two releases each, $P1_i$, $P1_j$, $P2_i$ and $P2_j$, respectively, where i < j. Assume package P1 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Assume P2 has maximum possible values of the properties or aspects c_1, c_2,...,c_n in release $P2_i$ and unchanged count d_1, d_2,...,d_n in release $P2_j$. If $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count \neq (d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count$, where count is property count or aspect count then $Stability_{P1} \neq Stability_{P2}$.

- **Property 2: A metric must preserve Representation Condition.** Let P1 and P2 be two packages with two releases each, $P1_i$, $P1_j$, $P2_i$ and $P2_j$, respectively, where i < j. Assume package P1 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Assume package P2 has maximum possible values of the properties or aspects c_1, c_2,...,c_n in release $P2_i$ and unchanged count d_1, d_2,...,d_n in release $P2_j$. If $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count > (d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count$, where count is property count or aspect count then $Stability_{P1} > Stability_{P2}$.

- **Property 3: Contribution of each unit of an entity's attribute must be same.** Let P1 and P2 be two packages with two releases each, $P1_i$, $P1_j$, $P2_i$ and $P2_j$, respectively, where i < j. Assume package P1 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Assume package P2 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P2_i$ and unchanged count b_1, b_2,...,$b_n + 1$ in release $P2_j$, then $Stability_{P2} = Stability_{P1} + 1/((a_1 + a_2 + ... + a_n) \times count)$.

- **Property 4: Measurement of different entities can be same.** Let P1 and P2 be two packages with two releases each, $P1_i$, $P1_j$, $P2_i$ and $P2_j$, respectively, where i < j. Assume package P1 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Assume pacakge P2 that it has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P2_i$ and unchanged count b_1, b_2,...,b_n in release $P2_j$, then $Stability_{P1} = Stability_{P2}$.

The proposed package level stability metrics satisfy all four properties of Kitchenham et al. framework [1] and hence these metrics are theoretically valid.

4.1.2 Hassan [2] Framework

Hassan [2] proposed seven mathematical properties for architecture stability. We can apply same properties to validate package stability because package also rep-

resent architecture upto some extent. Below is the evaluation of our metrics using these properties.

- **Property 1 Non-negativity:** According to this property the stability should be greater than or equal to zero. In proposed metrics, calculation requires operations like ratio, sum and average using unchanged counts and maximum possible change counts. Value of count is always greater than or equal to zero, so result of proposed metrics cannot be negative.

- **Property 2 Normalization:** Normalization property requires value of stability between a bounded interval. In our metrics we have selected interval [0 , 1], where value '1' means that package is completely stable and value '0' means that package is completely unstable. We are taking ratios between unchanged count and maximum possible change count. As we know unchanged count is always less than or equal to maximum possible change count, so stability of package using our metrics will always be between interval [0 , 1].

- **Property 3 Null Value:** Null value property says that architecture stability is null if all the inter package connections are changed. In our case null value means that package stability should be null if all properties and connections explained in section 4 are changed. In calculations for PCS, IPIS and EPIS, the numerator is the unchanged count. So if unchanged count of all properties and connections is zero then package stability will be zero or in other words null. Suppose we have a packages P with two releases

P_i and P_j, where i < j. Assume package P has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release P_i and unchanged count b_1=0, b_2=0,...,b_n=0 in release P_j, then $Stability_P = (b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count = 0$, where count is property count or aspect count.

- **Property 4 Maximum Value:** This property says that stability will have maximum value if there are only additions but no modification or deletion. In our metrics, unchanged count will be equal to maximum possible change count if there is no modification or deletion between two versions. This will result stability of each property to 1, so stability of package will have maximum value '1'. Suppose we have a packages P with two releases P_i and P_j, where i < j. Assume package P has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release P_i and unchanged count a_1, a_2,...,a_n in release P_j, then $Stability_P = (a_1/a_1 + a_2/a_2 + ... + a_n/a_n)/count = 1$, where count is property count or aspect count.

- **Property 5 Transitivity:** According to this property consider we have three entities X,Y and Z. If $Stability_X < Stability_Y$ and $Stability_Y < Stability_Z$ then $Stability_X < Stability_Z$. To prove this suppose we have three packages P1, P2 and P3 with two releases each, $P1_i$, $P1_j$, $P2_i$, $P2_j$, $P3_i$ and $P3_j$ where i < j. Assume package P1 has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Assume same for P2 that it has maximum possible values of the properties or aspects c_1, c_2,...,c_n in release $P2_i$ and

unchanged count d_1, d_2,...,d_n in release $P2_j$. For P3 also assume that it has maximum possible values of the properties or aspects e_1, e_2,...,e_n in release $P3_i$ and unchanged count f_1, f_2,...,f_n in release $P3_j$. If $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count < (d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count$ and $(d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count < (f_1/e_1 + f_2/e_2 + ... + f_n/e_n)/count$ then $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count < (f_1/e_1 + f_2/e_2 + ... + f_n/e_n)/count$, where count is property count or aspect count. So it means using our metrics if $Stability_{P1} < Stability_{P2}$ and $Stability_{P2} < Stability_{P3}$ then $Stability_{P1} < Stability_{P3}$.

- **Property 6 Change Impact:** This property states that suppose we have an entity 'X' and if its unchanged count in version j with respect to version i is less than its unchanged count in version k with respect to version j than entity 'X' stability in version j with respect to version i is less then its stability in version k with respect to version j, provided that its maximum change count from version i to version j is not less than its maximum change count from version j to version k. To prove that our metrics hold this property, suppose we have a package P with three releases, P_i, P_j and P_k, where $i < j < k$. Assume package P has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Also assume that package P has maximum possible values of the properties or aspects c_1, c_2,...,c_n in release $P2_j$ and unchanged count d_1, d_2,...,d_n in release $P2_j$. If $b_1 < d_1$, $b_2 < d_2$, ... , $b_n < d_n$, provided

$a_1 \not< c_1$, $a_2 \not< c_2$, ... , $a_n \not< c_n$ then $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count < (d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count$, where count is property count or aspect count. So $stability_P$ in version j with respect to i will be less than $stability_P$ in version k with respect to j.

- **Property 7 Package Cohesion Impact:** According to this property, suppose we have an entity 'X' and if its unchanged count in version j with respect to version i is greater than its unchanged count in version k with respect to version j then entity 'X' stability in version j with respect to version i is greater than its stability in version k with respect to version j, provided that its maximum change count from version i to version j is not greater than its maximum change count from version j to version k. Author proposed this only for architecture stability so it was focused only on intra-package connections (package cohesion), but we will validate our all metrics against this property. To prove that our metrics hold this property, suppose we have a package P with three releases, P_i, P_j and P_k, where $i < j < k$. Assume package P has maximum possible values of the properties or aspects a_1, a_2,...,a_n in release $P1_i$ and unchanged count b_1, b_2,...,b_n in release $P1_j$. Also assume that package P has maximum possible values of the properties or aspects c_1, c_2,...,c_n in release $P2_j$ and unchanged count d_1, d_2,...,d_n in release $P2_j$. If $b_1 > d_1$, $b_2 > d_2$, ... , $b_n > d_n$, provided $a_1 \not> c_1$, $a_2 \not> c_2$, ... , $a_n \not> c_n$ then $(b_1/a_1 + b_2/a_2 + ... + b_n/a_n)/count > (d_1/c_1 + d_2/c_2 + ... + d_n/c_n)/count$, where count is property count or aspect count. So $stability_P$

in version j with respect to i will be greater than stability_P in version k with respect to j.

So our proposed metrics (PSM, PCS, IPIS and EPIS) are also valid according to the properties of Hassan [1] framework.

4.2 Empirical Validation

To empirically validate package stability metrics, we present four analysis: (1) first analysis explores the correlation between package stability metrics and maintenance effort; (2) second analysis explores the correlations among five existing stability metrics and three proposed package stability metrics; (3) third analysis applies principal component analysis [39] to explore the orthogonal dimensions within the set of stability metrics to confirm that our package level stability metrics indeed contribute new information and provide evidence that the new metrics better captures package level stability; and (4) forth analysis applies linear regression analysis for prediction of maintenance effort to evaluate performance of five existing stability metrics and four proposed package stability metrics.

4.2.1 Software Systems and Metrics

- **Data collection:** To collect open source projects as data for our analysis, we applied following guidelines:
  - Software type should be generic. By software type we mean domain for which software is developed for. Software types from which we

collected our data are desktop applications, game, development tool (ETL tool), graphical designing tool and enterprise solution. From table 4.1 provide details about type of open source softwares.

- Software should be of different sizes. From table 4.2 provide details about the size of open source softwares.
- Selected softwares should be popular among practitioners and continuously updated.
- Selected software should be part of empirical studies from literature.

We have selected five open source software systems from different domains: Buddi - a small desktop application [42], JHotDraw - a graphical tool for technical drawing [43], KolMafia - an online adventure game [44], Talend - Extract, Transform, Load (ETL) tool for database systems [45] and University Timetabling System - an enterprise software [46]. Table 4.1 presents an overview of five open source software systems. We have used four different versions of each open source software; where there is at least one year difference between release dates of individual versions. As a result, we have collected three stability measurements for each package of five open source software systems.

Table 4.2 presents descriptive statistics of five open source systems in term of number of packages, classes and lines of code. Values of mean, minimum, maximum and standard deviation show that packages used in our experiments are dynamic in terms of size (number of classes and lines of code).

'Buddi' system consists of 27,28,28 and 29 packages across four differnet versions of the system. Four different versions of 'JHotDraw' consists of 39, 47, 63 and 66 packages, respectively. 'KolMafia' consists of 114, 114, 116 and 116 packages across four different versions of the game. Similarly, four different versions of the 'Talend' system consists of 108, 109, 107 and 109 packages, respectively. Finally, 'Unitime' consists of 64, 72, 106 and 124 packages across four different versions of the system. In total, our empirical validation experiments contain input of 1586 packages with 23935 classes and 10871597 lines of code.

Table 4.1: Selected Open Source Software

	Name	Type	Detail
1	Buddi[42]	Small desktop budgeting application	Buddi is a simple budgeting program targeted for users with little or no financial background. It is a small desktop application.
2	JHotDraw[43]	Graphical Tool	JHotDraw is a Java GUI framework for technical and structured Graphics.
3	KolMafia[44]	Game	KoLmafia is a cross-platform desktop tool, which interfaces with the online adventure game, Kingdom of Loathing.
4	Talend[45]	ETL Tool	Talend is data integration tool which makes ETL easy from any data source to almost any analytics or operational tools.
5	UniTime[46]	University Timetabling System	UniTime is a comprehensive educational scheduling system that supports developing course and exam timetables, managing changes to these timetables, sharing rooms with other events, and scheduling students to individual classes.

- **Metric Selection:** In empirical validation, we adopt Li and Henry's maintenance effort measurement definition [47]: 'Maintenance effort metrics cal-

Table 4.2: Selected Open Source Software size statistics

		Package vise Class Count Statistics					Package vise Line of Code Statistics				
Project Name	Package Count	Class Count	Mean	Min	Max	Standard Deviation	Line Of Code	Mean	Min	Max	Standard Deviation
Buddi 3.2.2.6	27	213	7.89	1	43	8.64	25314	937.56	22	3672	793.14
Buddi 3.4.0.0	28	239	8.54	1	46	9.52	28594	1021.21	23	4202	883.84
Buddi 3.4.1.1	28	241	8.61	1	46	9.51	29201	1042.89	23	4252	893.45
Buddi 3.4.1.14	29	245	8.45	1	46	9.42	29887	1030.59	23	4339	903.54
JHotDraw 7.1	39	441	11.31	1	144	23.47	93224	2390.36	68	25960	4305.47
JHotDraw 7.2	47	576	12.26	1	154	23.58	123333	2624.11	27	30806	4704.54
JHotDraw 7.4.1	63	584	9.27	1	45	9.54	124888	1982.35	27	14871	2562.90
JHotDraw 7.6	66	614	9.30	1	45	9.87	134759	2041.80	45	15347	2621.30
KoLmafia 16.5	114	2156	19.08	1	202	30.48	661351	5801.32	51	82832	10823.01
KoLmafia 16.9	114	2174	19.24	1	203	31.00	673848	5910.95	51	85754	11158.58
KoLmafia 17.2	116	2308	20.07	1	209	34.01	713570	6151.47	51	89652	11617.86
KoLmafia 17.4	116	2331	20.27	1	213	34.88	728162	6277.26	51	94619	12100.68
Talend 5.6.0	108	1371	12.81	1	58	11.35	1449852	13424.56	41	63604	16978.57
Talend 5.6.2	109	1394	12.91	1	58	11.35	1495508	13720.26	41	64888	17391.61
Talend 6.1.1	107	1409	13.29	1	64	11.63	1591136	14870.43	41	67353	18510.15
Talend 6.2.1	109	1406	13.02	1	63	11.46	1596927	14650.71	41	67556	18721.51
Unitime 32	64	1338	21.24	1	279	46.02	275986	4312.28	78	47608	7999.06
Unitime 33	72	1409	19.85	1	281	43.88	316511	4395.99	78	48511	7842.26
Unitime 34	106	1651	15.72	1	307	38.91	369246	3483.45	42	46166	6463.81
Unitime 35	124	1835	14.92	1	317	37.33	410300	3308.87	41	46284	6285.93

culates effort in term of total added, deleted or modified line of code'. We use the Li and Henry's [47] maintenance effort metrics to explore the correlation between proposed metrics and maintenance effort. We select Martin's package instability metric [27], package stability metrics based on system design instability (SDI) [16] metric definition, package stability metrics based on relationship based similarity metric (RBMS) [19] definition, package stability metrics based on core calls instability (CCI) [20] definition and package stability metrics based on architecture stability metric (ASM) [21] metric definition to analyze the correlations among five existing stability metrics and proposed metrics. Furthermore, we use these five existing metrics along with proposed metrics to apply principal component analysis [39] in order to explore orthogonal dimensions within this set of stability metrics. Finally we performed prediction analysis using linear regression by using Li and Henry's [47] maintenance effort metrics as dependent variable and, five existing stability metrics and three proposed metrics as independent variables. Below are list of existing metrics that we used in our empirical validation:

- **Maintenance effort[47]:** It calculates the total count of added, deleted and updated lines of codes to measure maintenance effort.
- **Martin's package instability metric [27]:** It measure the ratio between efferent coupling and sum of efferent + afferent coupling, in order to calculate package instability.
- **System Design Instability (SDI) [16]:** It measures the number of

changes in the name classes and take ratio with total number of classes to measure system instability.

- **Relationship Based Similarity Metric (RBMS)[19]:** It measure the change in inheritance relationship and take ratio with total inheritance relationships to measure architecture stability.
- **Core Calls Instability (CCI) [20]:** It measures the change in calls between packages (external calls) and inside package (internal calls) and take ratio with total calls.
- **Architecture Stability Metric (ASM) [21]:** It measure the unchanged inter-packages calls and take ratio with total external calls.

Table 4.3 contains descriptive statistics of total eight stability metrics and maintenance effort. Standard variation, mean, minimum and maximum values shows that data is dynamic and spread across different ranges. Whereas skewness values show that all metrics do not follow a normal distribution and have non-parametric nature. Our all four proposed metrics (PCS, IPIS and EPIS) have negative skew distribution because most of the contents and interactions of packages remain same. Hence proposed metrics does not follow a normal distribution and has non-parametric nature. Existing stability metrics ASM and RBSM have negative skew distribution because most of the packages are stable. Hence ASM and RBSM does not follow a normal distribution and has non-parametric nature. While existing instability metrics SDI and CCI have positive skew distribution because most of the

packages retain their original form. Hence SDI and CCI has non-parametric nature and does not follow a normal distribution. Whereas maintenance effort follows a positive skew distribution, hence it also does not follow a normal distribution and does not share non-parametric nature. Package instability follows a weak negative skew distribution, hence we can say that it does not follow a normal distribution and has non-parametric nature.

Table 4.3: Descriptive statistics of stability metrics and maintenance effort.

	Metric	Min	Max	Mean	Std. Deviation	Skewness
1	PCS	0.07	1	0.91	0.12	-2.12
2	IPIS	0	1	0.98	0.10	-7.30
3	EPIS	0	1	0.97	0.11	-6.01
4	PSM	0.03	1	0.94	0.10	-4.11
5	SDI	0	1	0.05	0.15	4.28
6	ASM	0	1	0.97	0.13	-5.38
7	RBSM	0	1	0.94	0.19	-3.69
8	CCI	0	9.50	0.05	0.40	18
9	Martin's Instability	0	1	0.39	0.43	0.42
10	Maintenance	0	30620	948.34	2397.94	4.89

- **Software Tools:** In our empirical validation, we used three software tools Eclipse, Matlab, and Knime [48]. Eclipse is used to develop our custom Java tool to automate metrics measurements. Matlab is used for correlation analysis, principal component analysis, and regression analysis. Whereas we used Knime for the statistical analysis reported in table 4.3 for identification of data nature and distribution.

4.2.2 Software Stability Metric Tool

In our empirical validation, we developed a new Java tool to automate package stability measurement for our proposed metrics, five existing stability metrics and maintenance effort using JavaParser[49] library to parse source code of selected open source. Our tool analyzes Java source code of different versions of open source software, extract the required information to calculate proposed package level stability metrics.

High Level Class Diagram

High level class diagram of our custom developed software stability metric tool is available in Figure 4.1. Two classes "Parser" and "Java Parcer Library" are backbone of our tool. Purpose of "Parser" class is to provide bridge between java parser and classes that contain logic for metric calculation. While "Java Parser Library" is actual java parser library with list of parser classes. Main logic to calculate metric is implemented in "Project", "Package", "Class" and "Interface" classes. While "Variable", "Function" and "Connections" classes are used as data classes. Tool takes the directory path of two projects as input and first parse both project in two instances of project class with whole hierarchy of packages, classes, interfaces, variables, function and connections. After parsing, metric tool compare properties of two projects and calculate different metric values according to their definitions.

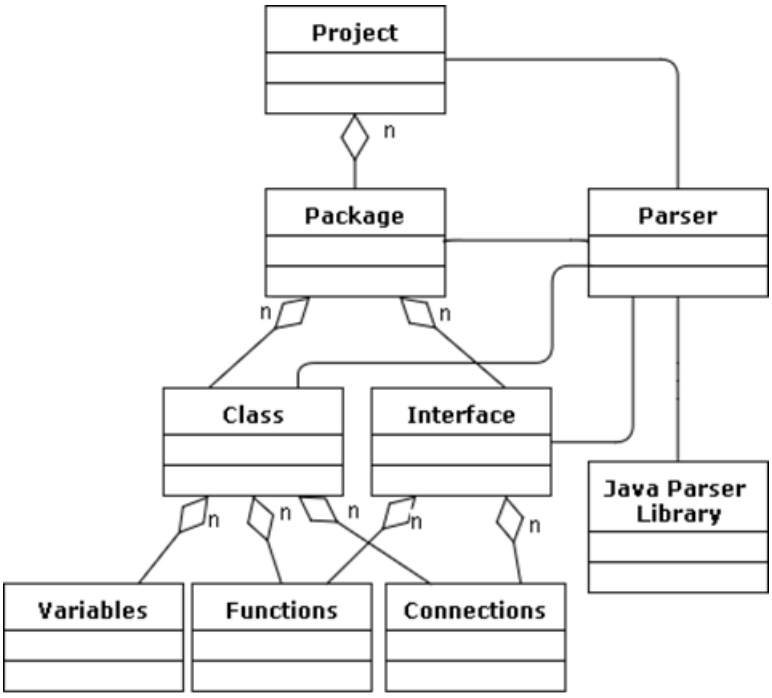


Figure 4.1: High Level Class Diagram of developed Software Stability Tool

Activity Diagram

Activity diagram of our custom developed software stability metric tool is available in Figure 4.2. Our custom developed metric tool performs below task in order to calculate stability metrics.

- **Traverse Project Directory:** As input tool require paths of two version of a software. As first stage, our tool parses the whole directory of both projects. It parses all the packages and files in those packages with extension ".java". It creates a list of packages objects with file lists in them.

- **Parse Java Files:** In second step, tool parses each file contents and extracts list of classes and interfaces. After that it creates list of objects of

parsed classes and interfaces, and then initialized the package object class and interface list property. Then tool parses each class and interface and extracts their variables, functions and connections.

- **Set Properties:** This step is called pre-processing. In this step our tool refine the data by defining and setting information for variables, functions and connections. For variable, it sets properties like access level, data type and name. For function it sets properties like return type, access level, function name and function's parameters properties. For connections, it sets properties like connection type, name of the class or interface the object is connected to. In last step, tool removes the duplicate connections.

- **Metric Measurement:** In this step, tool compares the list packages of both projects, then it compares the classes/interfaces, then it compares variables, functions and connections. As a result a comparison tool calculates the values of unchanged properties count, deleted properties count, added properties count and total maximum possible change count. At the end tool uses counts to calculate different metrics according to definitions.

- **Export Data:**In final step tool exports the calculated metric values with list of packages.

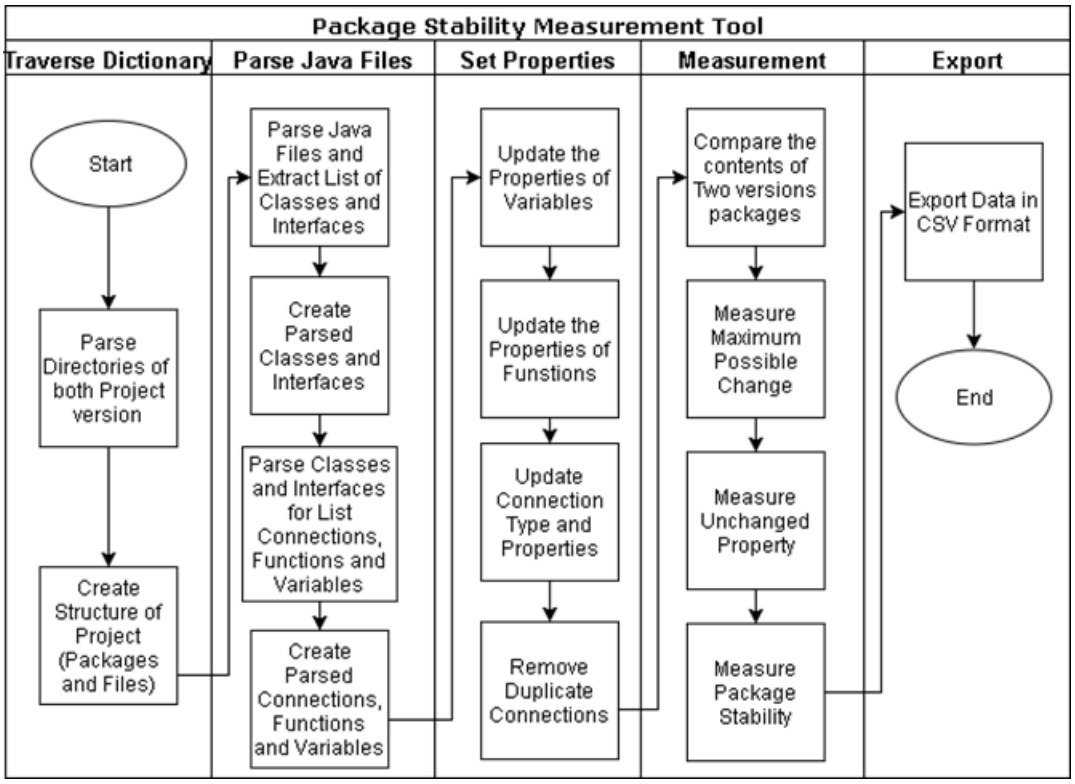


Figure 4.2: Activity Diagram of developed Software Stability Tool

4.2.3 Correlation with Maintenance Effort

Maintainability is one important property of design as software evolve to adapt changes in user requirements and operational environments [5]. Software stability contributes towards maintainability quality attribute and is one of the measures to obtain maintainable software. Stable software tends to minimize changes, improve maintainability and as a result help reduce maintenance effort [6, 7, 8]. We used correlation to analyze relationship between proposed package stability metrics with maintenance effort. We decided to use Spearman rank order correlation coefficient over Pearsons correlation coefficient [50] because data has

non-parametric nature and does not follow normal distribution as explained in section 4.3.1 using skewness values from table 4.2. We hypothesis the relationship between proposed package stability metrics and maintenance effort [47] as follows:

Null hypothesis: There is no significant association between the PCS metrics with maintenance effort.

Package content stability (PCS) has spearman rank order coefficient -0.83 and the p-value less than 0.5, as shown in Table 4.4. Hence, we reject the null hypothesis and conclude that PCS metric has a strong negative correlation with maintenance effort. Hence, we conclude that an increase in PCS will reduce maintenance effort.

Null hypothesis: There is no significant association between the IPIS metrics with maintenance effort.

The Spearman rank order is -0.29 and the p-value for intra-package interaction stability (IPIS) is less than 0.05, as shown in Table 4.4. Hence, we reject the null hypothesis and conclude that IPIS metric has a weak negative correlation with maintenance effort. Hence, we conclude that an increase in IPIS will reduce maintenance effort. Association is weak because IPIS considers changes in only those lines of code, which contribute towards intra-package interactions.

Null hypothesis: There is no significant association between the EPIS met-

rics with maintenance effort.

The Spearman rank order is -0.38 and the p-value for inter-package interaction stability (EPIS) is less than 0.05, as shown in Table 4.4. Hence, we reject the null hypothesis and conclude that EPIS metric has a moderate negative correlation with maintenance effort. Hence, we conclude that an increase in EPIS will reduce maintenance effort. Association is not strong because IPIS considers changes in only those lines of code, which contribute towards inter-package interactions.

Maintenance effort association with existing stability metric: We also performed correlation analysis between maintenance effort and package stability based on five exiting stability metric as shown in Table 4.4 and Figure 4.3. Package Content Stability (PCS) has stronger association with maintenance effort as compare to other stability all metrics. Inter-Package Interaction Stability (EPIS) has better association with maintenance effort as compared to any inter-package interaction based stability metric (ASM, RBSM, CCI and Martin Instability). Intra-Package Interaction Stability (IPIS) has weak association with maintenance effort, but it is the only stability metric available which covers all intra-package interactions. Hence our proposed stability metrics has better relationship with maintenance effort as compared to existing stability metrics.

Table 4.4: Correlation Analysis with Maintenance Effort. P Value less that 0.00005 is replace with 0

	Stability Metric	Spearman Rank Cofficient	P Value
1	PCS	-0.83	0
2	IPIS	-0.29	0
3	EPIS	-0.38	0
4	SDI	0.45	0
5	ASM	-0.35	0
6	RBSM	-0.35	0
7	CCI	0.36	0
8	Martin Instability	-0.04	0.20

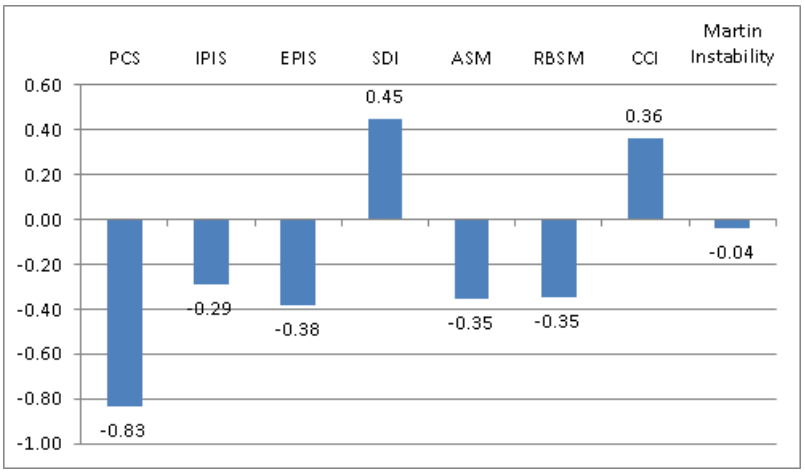


Figure 4.3: Correlation Analysis with Maintenance Effort.

4.2.4 Comparison with existing stability metric

In this section, we explore the correlations between proposed metrics suite and five existing stability metrics, namely, Martin's package instability metric [27], package stability metrics based on system design instability (SDI) [16] metric definition, package stability metrics based on relationship based similarity metric (RBMS) [19] definition, package stability metrics based on core calls instability

(CCI) [20] definition and package stability metrics based on architecture stability metric (ASM) [21] metric definition. We use Spearman rank order correlation method. Table 4.5 shows the results of correlation analysis between proposed metric suite and existing stability metrics.

- **Package Stability Based on SDI [16]** SDI also calculates the stability of a package contents by measuring changes in name of classes only and ignore other content aspects. Hence it has a weak correlation with PCS. Interestingly SDI has a better correlation with IPIS and EPIS as compared to PCS. The reason behind is that changes like the addition of new classes and deletion of existing classes, increase changes in class name and which affect the interactions between classes.

- **Package Stability Based on ASM [21]** ASM has a moderate correlation with EPIS because EPIS is also inter-package interactions based stability metric. The difference is that EPIS considers all ten types of inter-package interactions as explained in table 4.2. ASM has a weak correlation with PCS and IPIS because these metrics measure changes in contents and intra-package interactions respectively.

- **Package Stability Based on RBSM [19]** RBSM has a moderate correlation with EPIS because EPIS is also inter-package interactions based stability metric. The difference is that EPIS considers all ten types of inter-package interactions as explained in table 4.2. RBSM has a weak correlation

with PCS and IPIS because these metrics measure changes in contents and intra-package interactions respectively.

- **Package Stability Based on CCI [20]** CCI has a moderate correlation with EPIS because EPIS is also inter-package interactions based stability metric. The difference is that EPIS considers all ten types of inter-package interactions as explained in table 4.2. CCI has a weak correlation with PCS and IPIS because these metrics measure changes in contents and intra-package interactions respectively.

- **Package Instability [27]** Package instability metric has no correlation with PCS and IPIS because pValue is very high and Spearman rank order coefficient value is very low. Whereas Package instability metric has very weak negative correlation with EPIS and other existing stability metrics (ASM and RBSM). Whereas it has very weak positive correlation with instability metrics (SDI and CCI). Hence package instability has either no association with proposed metrics PCS and IPIS. Whereas packages instability has very weak correlation with metric that calculates inter-package interaction. The relationship is weak because package instability does not measure changes.

4.2.5 Principal Component Analysis

Principal Component Analysis (PCA) [39, 40] is a statistical tool that uses orthogonal transformation to identify and analyze underline relations and dimensions of the input parameters. Here it is used to understand the underlying orthogonal

Table 4.5: Correlation Analysis (Comparison with existing stability metrics). Note: R is Spearman Rank Order Coefficient and P less than 0.0005 is replaced with 0

Stability Metrics	PCS	IPIS	EPIS	SDI	ASM	RBSM	CCI
IPIS	R=0.26, P=0						
EPIS	R=0.36, P=0	R=0.46, P=0					
SDI	R=-0.31, P=0	R=-0.42, P=0	R=-0.47, P=0				
ASM	R=0.31, P=0	R=0.3, P=0	R=0.65, P=0	R=-0.34, P=0			
RBSM	R=0.31, P=0	R=0.35, P=0	R=0.59, P=0	R=-0.33, P=0	R=0.86, P=0		
CCI	R=-0.32, P=0	R=-0.35, P=0	R=-0.64, P=0	R=0.37, P=0	R=-0.98, P=0	R=-0.85, P=0	
Martin Instability	R=-0.01, P=0.67	R=-0.06, P=0.04	R=-0.29, P=0	R=0.19, P=0	R=-0.33, P=0	R=-0.32, P=0	R=0.33, P=0

dimensions and the relationship of five existing stability metrics and proposed metrics. In addition, PCA also demonstrates that the proposed package stability metric captures new measurement dimensions. Our experiment shows that outliers do not affect final PCA results.

To perform principal component analyses, we used singular value decomposition (SVD) [51] algorithm. Table 4.6 shows the results of PCA with all eight PCs (Principal Component). It also shows the eigenvalues, their percentages, and the cumulative percentage. For every eight PCs, coefficients of five existing stability metrics, PCS, IPIS and EPIS are also presented in Table 4.6. Coefficients indicate which are the influential metrics contributing to the captured dimension. Based of recommendations from literature and close coefficient values of PC 6 and PC 4, We decided to apply 0.45 cut on coefficients. We have removed PSM from PCA analysis because our major target was to study new dimensions of PCS, IPIS, and EPIS, and PSM is derived from them, hence it can affect their results. For each PC (principal component) analysis based on coefficients are follows:

- **PC 1:** Package instability metric is the only influential metric for this PC with coefficient value 0.97. It calculates instability of package by using the afferent coupling and efferent coupling. It does not measure changes between two versions of package. Hence it is unique from all other stability metric and captures different dimension.

- **PC 2:** RBSM is major influential metrics for this PC with coefficient values -0.57. Whereas CCI, SDI and ASM also have weak influence on this PC.

- **PC 3:** RBSM and SDI are major influential metrics for this PC with coefficient value 0.49 and 0.61 respectively.

- **PC 4:** ASM, RBSM and CCI are major influential metrics for this PC with coefficient values 0.52, 0.61 and 0.47 respectively.

- **PC 5:** PCS and SDI are major influential metrics for this PC with coefficient values 0.73 and 0.63 respectively.

- **PC 6:** PCS, IPIS and EPIS are major influential metrics for this PC with coefficient values -0.48, 0.52 and 0.59 respectively.

- **PC 7:** IPIS and EPIS are major influential metrics for this PC with coefficient values 0.73 and -0.62 respectively.

- **PC 8:** ASM and CCI are major influential metrics for this PC with coefficient values 0.72 and 0.66 respectively.

The PCA results show that our proposed Package Content Stability (PCS) metric covers a new dimension as it is major significant factor in PC 5 and 6. This proves that PCS captures new functional (package content) stability aspects. IPIS and EPIS metrics also captures new measurement dimension as they are major significant factors in PC 6 and PC 7. This proves that both metrics covers new dimension of intra and inter package interactions (structural and behavioral) stability.

Table 4.6: Principal Component Analyses Results.

	PC 1	PC 2	PC 3	PC 4	PC 5	PC 6	PC 7	PC 8
Eigen Values	0.19	0.06	0.03	0.01	0.01	0.01	0	0
Percentage	60.26	19.40	8.93	3.84	3.14	2.51	1.30	0.61
Cum. Percentage	60.26	79.66	88.59	92.44	95.58	98.09	99.39	100
PCS	0	0.27	-0.36	0.17	**0.73**	**-0.48**	-0.05	0.01
IPIS	-0.01	0.21	-0.34	0.02	0.14	**0.52**	**0.73**	0.14
EPIS	-0.04	0.27	-0.25	-0.26	0.19	**0.59**	**-0.62**	-0.16
SDI	0.03	-0.35	**0.61**	-0.19	**0.63**	0.24	0.09	-0.01
ASM	-0.09	0.38	0.18	**-0.52**	-0.05	-0.16	-0.04	**0.72**
RBSM	-0.18	**0.57**	**0.49**	**0.61**	-0.02	0.15	-0.06	0.02
CCI	0.10	-0.42	-0.19	**0.47**	0.06	0.21	-0.27	**0.66**
Martin Instability	**0.97**	0.21	0.09	0.01	-0.02	0.01	-0.01	0

4.2.6 Regression Analysis

To evaluate and compare the ability of five existing stability metric and newly proposed metrics to predict package maintenance effort, we used linear regression [41]. This method is widely applied by many studies [52, 53] for prediction in software engineering. Almugrin et al. [53] used linear regression to predict package maintainability and testability. In linear regression, we use independent variables to explain and predict the dependent variable. In our analysis five existing stability metric and newly proposed metrics are independent variables and maintenance effort is a dependent variable. Data from four different version of five systems as explained in table 4.1 were collected and then combined into one file for analysis. Linear regression assumes that the relationship between independent variables and the dependent variable is approximately linear. Linear regression also requires data to follow a normal distribution and we know that our data

does not follow a normal distribution. There exist transformation to make data normally distributed such as logarithmic, square roots and inverse transforms. We applied square root function onto the data of five existing stability metrics, PSM metric suite, and maintenance effort to make is normally distributed.

To analyze the prediction ability of package stability metrics three regression analysis were performed, (1) first analysis explores the performance of content-based stability metric to predict maintenance effort individually; (2) second analysis presents the performance of package interaction based stability metric to predict maintenance effort individually; and (3) third analysis find the best possible combination of stability metric to maximize prediction results with less number of dependent variables. Tables 4.7, 4.8 and 4.9 presents the results of analysis (1), (2) and (3) respectively. In these tables R-Squared value is the accuracy of prediction, the Adjusted R-Squared value is the actual accuracy of prediction, pValue shows the significance of experiment and F-statistic shows the significance of the relationship between independent and dependent variable.

- **Content based stability metrics:** In this analysis, we used only contents based stability metrics individually and perform the linear regression to predict maintenance effort. PCS and SDI are content-based package stability metrics. Table 4.7 presents the results of this analysis. In comparison with SDI, PCS is producing good accuracy with adjusted R-squared value 0.378.

- **Interaction based stability metrics:** In this analysis, we used only in-

Table 4.7: Summary of the linear regression using content based stability metrics as independent variable.

	Stability Metrics	R-Squared	Adj. R-Squared	pValue	F-statistic
1	PCS	0.378	0.378	<0.00001	646
2	SDI	0.127	0.126	<0.00001	154

teractions based stability metrics individually as the independent variables and perform the linear regression to predict maintenance effort. IPIS, EPIS, Package Instability, ASM, RBSM and CCI calculates package stability using interactions. Table 4.8 presents the results of this analysis. In comparison with other interaction-based stability metrics, IPIS and EPIS are producing good results. Adjusted R-squared value of IPIS is 0.0367 whereas the adjusted R-squared value of EPIS is 0.0325. As compared to content-based stability metrics these values are low because most of the packages remain stable from the interaction point of view but the maintenance effort measures change in each line of code whether it contributes toward interaction or not.

Table 4.8: Summary of the linear regression using package interactions based stability metrics as independent variable.

	Stability Metrics	R-Squared	Adj. R-Squared	pValue	F-statistic
1	IPIS	0.0376	0.0367	<0.00001	39.1
2	EPIS	0.0335	0.0325	<0.00001	34.6
3	ASM	0.026	0.0251	<0.00001	28.4
4	RBSM	0.0282	0.0273	<0.00001	30.8
5	CCI	0.0213	0.0204	<0.00001	30.1
6	Package Instability	0.0182	0.0173	<0.00001	19.7

- **Best combination of stability metrics:** In this analysis, we presents the best possible combinations of stability metric that can produce good results. Table 4.9 presents the results of this analysis.
  - Combination 1: In row 1 by using proposed metric as independent variable we get 0.399 accuracy.
  - Combination 2: While in row 2 by using all existing stability metrics as independent variable, accuracy to predict maintenance effort is 0.178 accuracy.
  - Combination 3: If we use all five existing stability metrics with proposed metric as independent variable, accuracy to predict maintenance effort accuracy improves to 0.437.
  - Combination 4: We tried different possible combination and come up with the best combination of five independent variable: PCS, IPIS, EPIS, SDI and Package Instability. This combination predicts maintenance effort with accuracy of 0.43.

Table 4.9: Summary of the linear regression using best combinations.

	Description	Dependent Variables	R-Sq.	Adj. R-Sq.	pValue	F-stat
1	Proposed Metrics	PCS, IPIS and EPIS	0.401	0.399	<0.00001	236
2	Five Existing Metrics	SDI, ASM, RBSM, CCI and Instability	0.181	0.178	<0.00001	58.6
3	Combined	All	0.44	0.437	<0.00001	119
4	Proposed Combination	PCS, IPIS, EPIS, SDI,and Instability	0.432	0.43	<0.00001	161

The linear regression results show that our proposed metrics improved the prediction accuracy of maintenance effort. In table 4.9, we can see that in row 2 using all existing metric the maintenance effort prediction accuracy is 0.178. But when we introduce proposed metric suite in row 3 the accuracy is improved to 0.437. But using all stability metrics will be very costly so we proposed a combination of five stability metrics that include PCS, IPIS, EPIS, SDI and Package Instability which produce an acceptable accuracy of 0.43. Finally, we present the Linear regression model using these five metrics in table 4.10. In the model estimate is the corresponding coefficient, the standard error is expected error in estimate value and, tStat and pValue show the significance.

Table 4.10: Regression model of best possible combination.

	Estimate	Standard Error	tStat	pValue
(Intercept)	268.36	20.833	12.881	<0.00001
PCS	-125.62	14.337	-8.762	<0.00001
IPIS	18.887	9.9634	1.8956	0.058304
EPIS	23.252	10.693	2.1746	0.029895
SDI	27.573	4.0368	6.8305	<0.00001
Package Instability	-180.27	22.819	-7.8999	<0.00001

CHAPTER 5

COMPARISON OF REGRESSION AND CLASSIFICATION TO PREDICT PACKAGE MAINTAINABILITY

In this chapter we presents initial results of comparison analysis between regression algorithm and classification algorithm to predict package maintainability. We conducted four set of analysis for this comparison; (1) first analysis explores the correlation between five existing stability metrics, PSM metric suite, two package cohesion metrics, two package coupling metric, maintenance effort and future (next version) maintenance effort; (2) second analysis applies principal compo-

nent analysis to explore which metrics covers different directions and are unique from others; (3) third analysis applies three to predict package current maintainability and future (next version) maintainability; and (4) forth analysis applies six classification algorithms to predict package current maintainability and future (next version) maintainability. Bellow are the differences between analysis in this chapter and analysis in chapter 4.

- Analysis in this chapter are based on 26 open source softwares. We collected four version of each 26 open software to measure stability and maintenance.
- Analysis in this also contain input of two package cohesion and two package coupling metrics.
- This chapter also present prediction analysis of future maintenance effort.
- Major contribution of this is the comparison analysis between regression and classification algorithms.

5.1 Data Collection

We have selected 26 open source software systems from different domains for analysis in this chapter. We have used four different versions of each open source software; where there is at least one year difference between release dates of individual versions. As a result, we have collected three stability measurements, three maintenance effort measurements and two future maintenance effort measurements. Table 5.1 presents descriptive statistics of 4th version of each 26 open

source systems in term of number of packages, classes and lines of code. Values of mean, minimum, maximum and standard deviation show that packages used in our experiments are dynamic in terms of size (number of classes and lines of code). In total, experiments in this chapter contain input of 20954 packages with 169020 classes and 35270848 lines of code.

5.2 Metric Selection

In this chapter, we adopt Li and Henry's maintenance effort measurement definition [47]: 'Maintenance effort metrics calculates effort in term of total added, deleted or modified line of code' to predict package maintenance. We ignored Martin's package instability metric [27] because it did not generate good result in chapter 4. We select package stability metrics based on Li's class implementation instability (CII) [16] metric definition (PCII), package stability metrics based on system design instability (SDI) [16] metric definition (PSDI), package stability metrics based on class number of method stability [22] metric definition (PNomStab), package stability metrics based on class stability metric (CSM) [5] definition (PCSM) and package stability metrics based on architecture stability metric (ASM) [21] metric definition (PASM) to analyze the correlations among six existing stability metrics and PSM. We also used two package cohesion metric; Martin's Cohesion [27] and Component Cohesion of Vernazza et. al. [54]. We also used tow package coupling metrics; afferent coupling and efferent coupling [27].

Table 5.2 contains descriptive statistics of total nine stability metrics, two

Table 5.1: 26 Selected Open Source Software size statistics

			Package vise Class Count Statistics					Package vise Line of Code Statistics				
Software	Version	Package Count	Class Count	Min	Max	Mean	Std. Dev	Line Of Code	Min	Max	Mean	Std. Dev
Buddi	3.4.1.14	30	212	1	46	7.07	9.16	29887	23	4339	996.23	903.54
JHotDraw	7.6	66	553	1	39	8.38	8.77	134759	45	15347	2041.80	2621.30
KolMafia	17.4	115	2160	1	213	18.78	34.39	728162	51	94619	6331.84	12100.68
Talend	6.2.1	108	1392	1	58	12.89	11.34	1596927	41	67556	14786.36	18721.51
UniTime	Bundle 35	123	1725	1	317	14.02	37.36	410300	41	46284	3335.77	6285.93
Apache Camel	2.18.1	1602	13044	1	615	8.14	24.07	1357680	25	50471	847.49	2144.60
Apache Tomcat	9.0.0.M17	168	1162	0	54	6.92	8.72	275842	23	21673	1641.92	2627.85
JEdit	5.3.0	42	514	1	81	12.24	17.37	187041	65	30284	4453.36	7319.91
Mule	3.7.5	1015	5910	1	64	5.82	7.89	664365	12	13677	654.55	1057.39
Hadoop	2.7.3	742	5408	1	139	7.29	12.62	1367213	33	57580	1842.61	4310.51
Synapse	3.0.0	215	1152	1	123	5.36	9.97	175151	48	16102	814.66	1540.27
Apache Ant	1.10.0	48	365	1	86	7.60	14.18	60797	25	14022	1266.60	2457.94
Essence	0.75	26	124	1	17	4.77	4.81	22196	15	4582	853.69	1010.75
Gridsim	5.2	60	274	1	34	4.57	5.54	99488	148	20184	1658.13	2884.28
JavaGroups	3.6.5	13	117	0	27	9.00	8.56	32583	108	10895	2506.38	3327.34
Prevayler	2.02.006	39	156	1	14	4.00	2.79	11180	35	969	286.67	202.84
Super (Acelet Scheduler)	3	10	243	1	81	24.30	25.76	39997	67	13439	3999.70	3998.37
DBUnit	2.4.9	30	272	1	34	9.07	7.73	47301	115	5982	1576.70	1478.90
BCEL	6.0	12	376	1	205	31.33	57.41	55802	260	24362	4650.17	7137.46
Code Generation Library	3.1	18	159	2	26	8.83	7.70	20357	8	4878	1130.94	1236.49
Spring	4.3.6	822	5248	1	71	6.38	7.61	979993	24	17961	1192.21	1788.32
Java X11 Library	0.3	16	244	1	47	15.25	13.51	42228	6	11478	2639.25	3239.17
DrJava	r4668	28	489	1	68	17.46	18.29	143586	52	33038	5128.07	7074.89
Find bugs	3.0.0	77	975	1	186	12.66	28.32	208578	34	42130	2708.81	7047.55
Jasper reports	6.2.0	124	1992	1	195	16.06	26.29	455611	62	59180	3674.28	7978.47
Junit	4.9	28	121	1	11	4.32	3.13	11545	31	1439	412.32	399.31

package cohesion metrics, two package coupling metric, maintenance effort and future maintenance effort. Standard variation, mean, minimum and maximum values shows that data is dynamic and spread across different ranges. Whereas skewness values show that all metrics except two cohesion metrics, do not follow a normal distribution and have non-parametric nature. Only package cohesion metric; martin cohesion and component cohesion follows normal distribution.

Table 5.2: Descriptive statistics of stability, cohesion, coupling and maintenance effort metrics.

	Metric	Min	Max	Mean	Std. Dev	Skewness
1	PCS	0	1	0.955	0.098	-3.981
2	IPIS	0	1	0.990	0.080	-10.176
3	EPIS	0	1	0.949	0.175	-3.989
4	PSM	0	1	0.955	0.109	-4.119
5	PCII	0	1	0.040	0.115	4.923
6	PSDI	0	1	0.031	0.118	5.323
7	PNomStab	0	1	0.962	0.123	-4.986
8	PCSM	0.375	1	0.976	0.058	-4.051
9	PASM	0	1	0.944	0.197	-3.905
10	Martin Cohesion	0.003	3.875	0.657	0.382	0.619
11	Component Cohesion	0	1	0.443	0.442	0.352
12	Afferent Coupling	0	4274	11.338	97.237	26.115
13	Efferent Coupling	0	299	7.985	15.347	7.537
14	Maintenance	0	34123	170.742	873.310	14.963
15	Future Maintenance	0	34123	172.232	942.157	15.758

5.3 Metric Tool

I developed custom software stability metric tool for my experiments. Details of software stability metric tool are provided in section 4.3.2.

5.4 Software Tool

As explained in section 4.3.1, we used three software tools Eclipse, Matlab, and Knime [48]. Eclipse is used to develop our custom Java tool to automate metrics measurements. Matlab is used for correlation analysis, principal component analysis, regression analysis and clustering. Whereas we used Knime for the classification algorithms.

5.5 Correlation Analysis

We performed correlation analysis between 15 different metrics. These metrics are maintenance effort [47], future maintenance effort, PSM metric suite, PCII [16], PSDI [16], PNomStab [22], PCSM [5], PASM [21], Martin's Cohesion [27], Component Cohesion [54], afferent coupling and efferent coupling [27]. We used pearson rank correlation analysis because most of our data nature is non-parametric and does not follow normal distribution. Correlation analysis results are available in tables 5.3 and 5.4. Some analysis are listed below:

- Proposed metrics has better correlation with maintenance effort as compared to other stability metric.
- Proposed metrics has better correlation with future maintenance effort as compared to other stability metric.
- Most of stability metrics including Proposed metrics has weak negative correlation with afferent and efferent coupling. Where as instability metric has

weak positive correlation with afferent and efferent coupling.

- Most of stability metrics including Proposed metrics has weak positive correlation with martin cohesion and component cohesion. Where as instability metric has weak negative correlation with martin cohesion and component cohesion.

- Maintenance effort and future maintenance has weak positive correlation with afferent and efferent coupling.

- Maintenance effort and future maintenance has weak negative correlation with martin cohesion and component cohesion.

5.6 Principal Component Analysis

Principal Component Analysis (PCA) [39, 40] is a statistical tool that uses orthogonal transformation to identify and analyze underline relations and dimensions of the input parameters. Here it is used to understand the underlying orthogonal dimensions and the relationship between 13 different metrics; PSM metric suite, PCII [16], PSDI [16], PNomStab [22], PCSM [5], PASM [21], Martin's Cohesion [27], Component Cohesion [54], afferent coupling and efferent coupling [27]. We also performed PCA analysis for dimension reduction. We performed two PCA analyis; one with all 13 metrics and second with only stability metrics. Due to recommendation from literature and close coefficient values of PC 6, we applied coefficient cut at 0.45.

Table 5.3: Correlation analysis of package stability, cohesion, coupling and maintenance effort (Part 1). Note: P values less than 0.00001 are replaced by 0.

		PCS	IPIS	EPIS	PSM	PCII	PSDI	PNomStab	PCSM
1	PCS	R= 1, P= 1	R= 0.267, P= 0	R= 0.262, P= 0	R= 0.848, P= 0	R= -0.824, P= 0	R= -0.385, P= 0	R= 0.654, P= 0	R= 0.828, P= 0
2	IPIS	R= 0.267, P= 0	R= 1, P= 1	R= 0.267, P= 0	R= 0.266, P= 0	R= -0.246, P= 0	R= -0.361, P= 0	R= 0.291, P= 0	R= 0.228, P= 0
3	EPIS	R= 0.262, P= 0	R= 0.267, P= 0	R= 1, P= 1	R= 0.612, P= 0	R= -0.241, P= 0	R= -0.293, P= 0	R= 0.287, P= 0	R= 0.345, P= 0
4	PSM	R= 0.848, P= 0	R= 0.266, P= 0	R= 0.612, P= 0	R= 1, P= 1	R= -0.702, P= 0	R= -0.353, P= 0	R= 0.568, P= 0	R= 0.8, P= 0
5	PCII	R= -0.824, P= 0	R= -0.246, P= 0	R= -0.241, P= 0	R= -0.702, P= 0	R= 1, P= 1	R= 0.576, P= 0	R= -0.775, P= 0	R= -0.713, P= 0
6	PSDI	R= -0.385, P= 0	R= -0.361, P= 0	R= -0.293, P= 0	R= -0.353, P= 0	R= 0.576, P= 0	R= 1, P= 1	R= -0.654, P= 0	R= -0.311, P= 0
7	PNomStab	R= 0.654, P= 0	R= 0.291, P= 0	R= 0.287, P= 0	R= 0.568, P= 0	R= -0.775, P= 0	R= -0.654, P= 0	R= 1, P= 1	R= 0.598, P= 0
8	PCSM	R= 0.828, P= 0	R= 0.228, P= 0	R= 0.345, P= 0	R= 0.8, P= 0	R= -0.713, P= 0	R= -0.311, P= 0	R= 0.598, P= 0	R= 1, P= 1
9	PASM	R= 0.313, P= 0	R= 0.223, P= 0	R= 0.617, P= 0	R= 0.508, P= 0	R= -0.311, P= 0	R= -0.245, P= 0	R= 0.352, P= 0	R= 0.37, P= 0
10	Martin Cohesion	R= 0.108, P= 0	R= -0.036, P= 0	R= 0.066, P= 0	R= 0.126, P= 0	R= -0.155, P= 0	R= -0.159, P= 0	R= 0.136, P= 0	R= 0.111, P= 0
11	Component Cohesion	R= 0.203, P= 0	R= 0.07, P= 0	R= 0.119, P= 0	R= 0.199, P= 0	R= -0.25, P= 0	R= -0.224, P= 0	R= 0.236, P= 0	R= 0.214, P= 0
12	Afferent Coupling	R= -0.191, P= 0	R= -0.168, P= 0	R= -0.064, P= 0	R= -0.139, P= 0	R= 0.205, P= 0	R= 0.144, P= 0	R= -0.19, P= 0	R= -0.179, P= 0
13	Efferent Coupling	R= -0.286, P= 0	R= -0.136, P= 0	R= -0.251, P= 0	R= -0.288, P= 0	R= 0.306, P= 0	R= 0.208, P= 0	R= -0.334, P= 0	R= -0.308, P= 0
14	Maintenance	R= -0.834, P= 0	R= -0.284, P= 0	R= -0.269, P= 0	R= -0.71, P= 0	R= 0.906, P= 0	R= 0.56, P= 0	R= -0.725, P= 0	R= -0.779, P= 0
15	Future Maintenance	R= -0.45, P= 0	R= -0.188, P= 0	R= -0.122, P= 0	R= -0.351, P= 0	R= 0.451, P= 0	R= 0.293, P= 0	R= -0.383, P= 0	R= -0.417, P= 0

Table 5.4: Correlation analysis of package stability, cohesion, coupling and maintenance effort (Part 2). Note: P values less than 0.00001 are replaced by 0.

		PASM	Martin Cohesion	Component Cohesion	Afferent Coupling	Efferent Coupling	Maintenance	Future Maintenance
1	PCS	R= 0.313, P= 0	R= 0.108, P= 0	R= 0.203, P= 0	R= -0.191, P= 0	R= -0.286, P= 0	R= -0.834, P= 0	R= -0.45, P= 0
2	IPIS	R= 0.223, P= 0	R= -0.036, P= 0	R= 0.07, P= 0	R= -0.168, P= 0	R= -0.136, P= 0	R= -0.284, P= 0	R= -0.188, P= 0
3	EPIS	R= 0.617, P= 0	R= 0.066, P= 0	R= 0.119, P= 0	R= -0.064, P= 0	R= -0.251, P= 0	R= -0.269, P= 0	R= -0.122, P= 0
4	PSM	R= 0.508, P= 0	R= 0.126, P= 0	R= 0.199, P= 0	R= -0.139, P= 0	R= -0.288, P= 0	R= -0.71, P= 0	R= -0.351, P= 0
5	PCII	R= -0.311, P= 0	R= -0.155, P= 0	R= -0.25, P= 0	R= 0.205, P= 0	R= 0.306, P= 0	R= 0.906, P= 0	R= 0.451, P= 0
6	PSDI	R= -0.245, P= 0	R= -0.159, P= 0	R= -0.224, P= 0	R= 0.144, P= 0	R= 0.208, P= 0	R= 0.56, P= 0	R= 0.293, P= 0
7	PNomStab	R= 0.352, P= 0	R= 0.136, P= 0	R= 0.236, P= 0	R= -0.19, P= 0	R= -0.334, P= 0	R= -0.725, P= 0	R= -0.383, P= 0
8	PCSM	R= 0.37, P= 0	R= 0.111, P= 0	R= 0.214, P= 0	R= -0.179, P= 0	R= -0.308, P= 0	R= -0.779, P= 0	R= -0.417, P= 0
9	PASM	R= 1, P= 1	R= 0.072, P= 0	R= 0.152, P= 0	R= -0.109, P= 0	R= -0.343, P= 0	R= -0.336, P= 0	R= -0.202, P= 0
10	Martin Cohesion	R= 0.072, P= 0	R= 1, P= 1	R= 0.746, P= 0	R= 0.057, P= 0	R= -0.179, P= 0	R= -0.183, P= 0	R= -0.163, P= 0
11	Component Cohesion	R= 0.152, P= 0	R= 0.746, P= 0	R= 1, P= 1	R= -0.132, P= 0	R= -0.36, P= 0	R= -0.32, P= 0	R= -0.305, P= 0
12	Afferent Coupling	R= -0.109, P= 0	R= 0.057, P= 0	R= -0.132, P= 0	R= 1, P= 1	R= 0.206, P= 0	R= 0.269, P= 0	R= 0.279, P= 0
13	Efferent Coupling	R= -0.343, P= 0	R= -0.179, P= 0	R= -0.36, P= 0	R= 0.206, P= 0	R= 1, P= 1	R= 0.36, P= 0	R= 0.354, P= 0
14	Maintenance	R= -0.336, P= 0	R= -0.183, P= 0	R= -0.32, P= 0	R= 0.269, P= 0	R= 0.36, P= 0	R= 1, P= 1	R= 0.549, P= 0
15	Future Maintenance	R= -0.202, P= 0	R= -0.163, P= 0	R= -0.305, P= 0	R= 0.279, P= 0	R= 0.354, P= 0	R= 0.549, P= 0	R= 1, P= 1

Table 5.5 presents PCA analysis results of 13 different stability, coupling and cohesion metrics. Some analysis are as follows:

- PC 1: Afferent coupling has total influence on this principal component. Hence Afferent coupling measure totally different dimension.
- PC 2: Efferent coupling has total influence on this principal component. Hence Efferent coupling measure totally different dimension.
- PC 3 and 4: Martin cohesion and component cohesion share total influence on these two principal component. Hence martin cohesion and component cohesion measure same dimension but different from other metrics.
- Remaining Principal Components: We know that results of PCA are affected if one metric has high variance and we know from table 5.2 that variance of coupling metrics is too high and variance of cohesion metrics is also high as compare to stability metrics. So for stability metrics we have performed separate PCA analysis.

Table 5.5 presents our second PCA analysis results of that consist of only stability metrics. Some analysis are as follow:

- PC 1 and 3: PASM and EPIS both are inter-package interaction stability metric and share same dimension. Both have influence on PC 1 and 3.
- PC 2: This principal component is influence by four existing metrics PCII, PSI, PNomStab and PASM.

- PC 4: This principal component is influence by two metric of proposed metrics; PCS and IPIS. This proves that proposed metrics cover new dimension of package stability.

- PC 5, 6, 7 and 8: Remaining principal components are covered by more than on stability metrics. In PC 5, 6 and 8, proposed metrics (PCS and IPIS) share influence with existing stability metrics. This means that PCS and IPIS also has relation with existing stability metrics.

5.7 Prediction using Regression

To evaluate and compare the ability of 13 different package stability, cohesion and coupling metric to predict package maintenance effort and future maintenance effort we have perform regression analysis. We used linear regression [55] [56] , quadratic regression [57] and polynomial regression (degree 3 and 4) [57]. Linear regression assume lenear relationship between dependent and independent variable and we know that we data is not linear. In order to make our data linear, we have used square root function. Table 5.3 presents the results of maintenance effort prediction in the form of Adj. R-squared values. Whereas table 5.4 presents the results of future maintenance effort prediction in the form of Adj. R-squared values. Figure 5.1 provides details for both prdictions. Some analysis of maintenance effort prediction are as follow:

- Polynomial regression with degree 4 produce best results.

Table 5.5: Principal component analysis with coupling and cohesion metrics.

	PCA 1	PCA 2	PCA 3	PCA 4	PCA 5	PCA 6	PCA 7	PCA 8	PCA 9	PCA 10
Eigen Values	9458.926	231.586	0.259	0.068	0.062	0.036	0.013	0.007	0.005	0.004
Percentage	97.606	2.390	0.003	0.001	0.001	0	0	0	0	0
Cum. Percentage	97.606	99.995	99.998	99.999	99.999	100	100	100	100	100
PCS	0	0	0.007	0.007	0.191	0.282	0.018	0.702	-0.351	-0.226
IPIS	0	0	0.006	-0.004	0.109	0.135	0.055	0.459	0.686	0.527
EPIS	0	0	0.012	0.019	0.579	-0.212	0.774	-0.124	-0.061	0.041
PCII	0	0	-0.023	-0.012	-0.246	-0.455	0.129	0.274	0.132	-0.150
PSDI	0	0	-0.026	-0.010	-0.242	-0.443	-0.008	0.070	-0.437	0.677
PNomStab	0	0	0.025	0.011	0.266	0.479	-0.150	-0.299	-0.169	0.414
PCSM	0	0	0.003	0	0.097	0.080	-0.013	0.333	-0.402	0.123
PASM	0	-0.001	0.031	-0.016	0.644	-0.468	-0.598	0.030	0.059	-0.052
Martin Cohesion	0	0	0.638	0.768	-0.038	-0.028	-0.006	0.010	0.001	0.003
Component Cohesion	0	-0.008	0.768	-0.639	-0.031	-0.002	0.025	0.001	-0.009	0.001
Afferent Coupling	1	-0.021	0	0	0	0	0	0	0	0
Efferent Coupling	0.021	1	0.006	-0.005	0.001	0	0	0	0	0

Table 5.6: Principal component analysis without coupling and cohesion metrics.

	PCA 1	PCA 2	PCA 3	PCA 4	PCA 5	PCA 6	PCA 7	PCA 8
Eigen Values	0.063	0.037	0.013	0.007	0.005	0.004	0.001	0.001
Percentage	47.838	27.957	10.312	5.060	3.900	2.957	1.015	0.962
Cum. Percentage	47.838	75.795	86.107	91.167	95.067	98.023	99.038	100
PCS	0.193	0.279	0.021	0.704	-0.348	-0.227	0.008	-0.465
IPIS	0.110	0.134	0.054	0.457	0.687	0.528	-0.074	0.046
EPIS	0.573	-0.220	0.776	-0.122	-0.059	0.041	-0.006	-0.021
PCII	-0.251	-0.453	0.125	0.273	0.135	-0.151	0.774	0.028
PSDI	-0.248	-0.441	-0.007	0.072	-0.439	0.675	-0.145	-0.263
PNomStab	0.272	0.477	-0.146	-0.297	-0.172	0.414	0.611	-0.121
PCSM	0.098	0.079	-0.012	0.335	-0.400	0.122	0.025	0.835
PASM	0.643	-0.472	-0.597	0.029	0.056	-0.051	0	-0.016

- Accuracy of coupling metrics for maintenance effort prediction is very low.
- Accuracy of cohesion metrics for maintenance effort prediction is very low.
- Prediction accuracy of PSM stability metric is less as compared to accuracy of combined five existing stability metric. Still when combine with other stability metrics, PSM metric suite increase the accuracy with a good value.

Some analysis of future maintenance effort prediction are as follow:

- Polynomial regression with degree 4 produce best results.
- Coupling and cohesion metrics also produce good results in comparison.
- Although accuracy of PSM metric suite is low but when we add it to other metrics, it improves the accuracy. Hence PSM metric suite is covering new dimension and helping to increase accuracy.

Table 5.7: Maintenance Effort Prediction using Regression. Table contain adjusted R-squared values.

	Linear Regression	Quadratic Regression	Polynomial Regression with degree 3	Polynomial Regression with degree 4
All Metrics	0.499	0.61	0.64	0.652
Coupling Metrics	0.117	0.142	0.16	0.17
Cohesion Metrics	0.0833	0.15	0.181	0.192
All Stability Metric	0.435	0.526	0.535	0.549
PSM Metrics Suite	0.256	0.368	0.384	0.41
All Five Stability Metrics Other than PSM	0.409	0.479	0.5	0.51

Table 5.8: Future Maintenance Effort Prediction using Regression. Table contain adjusted R-squared values.

	Linear Regression	Quadratic Regression	Polynomial Regression with degree 3	Polynomial Regression with degree 4
All Metrics	0.265	0.305	0.331	0.343
Coupling Metrics	0.186	0.1533	0.1619	0.1703
Cohesion Metrics	0.121	0.1278	0.1724	0.1841
All Stability Metric	0.17	0.1575	0.1851	0.207
PSM Metrics Suite	0.0946	0.1026	0.1207	0.1306
All Five Stability Metrics Other than PSM	0.156	0.1379	0.1677	0.1876

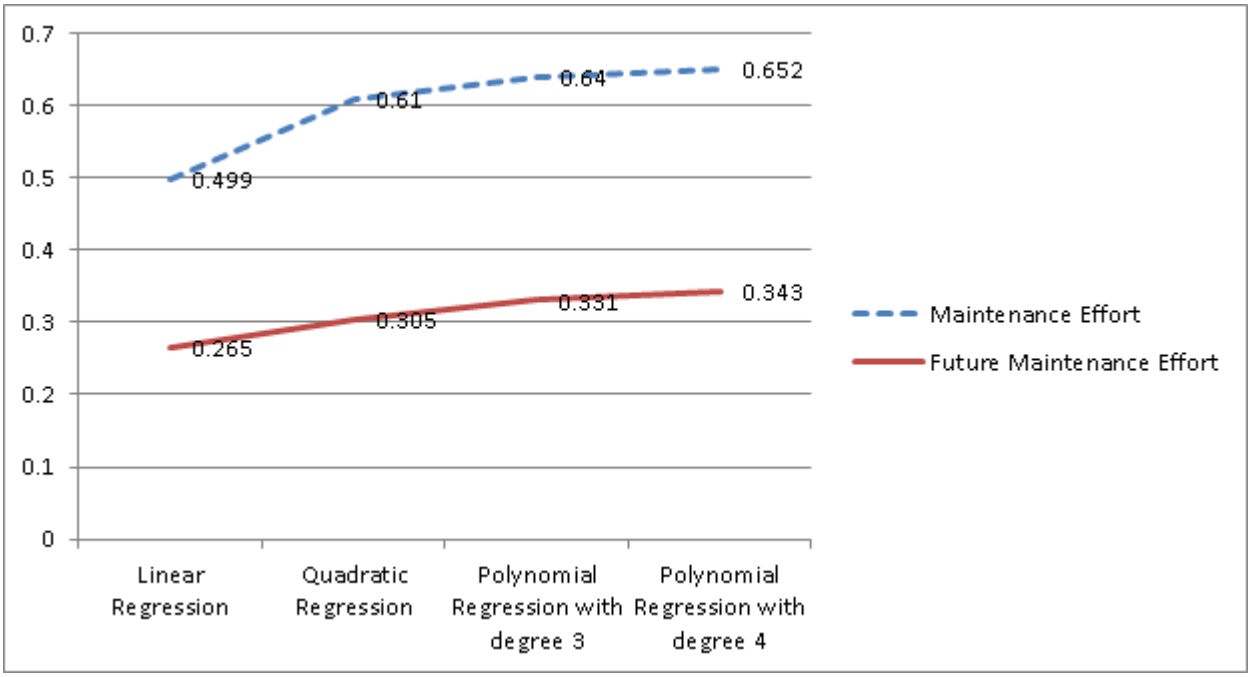


Figure 5.1: Prediction of maintenance effort and future maintenance effort using regression.

5.8 Prediction using Classification

Classification is use to predict discrete or categorical data. In our case we have continues data (maintenance effort and future maintenance effort). To convert our data into discrete data, we have use k-Mean clustering. We created 10 clusters using k-Mean clustering [58] and treated each cluster as separate class. Then we used basic classifies, Nave Bayes classifier [59] to analyze which stability metric produce better results. Table 5.9 presents the results of prediction in the form of accuracy percentages. After basic analysis and finding best combination, we used six classification technique and analyze there performance. Analysis of Nave Bayes classifier are bellow:

- Accuracy of Proposed metric suite is almost equal to accuracies of other stability metrics in prediction of maintenance effort.
- Accuracy of Proposed metric suite is better that the accuracies of other stability metrics in prediction of future maintenance effort.
- Interestingly when we combine different metrics, our accuracy reduces.
- K-mean cluster data using distances and does not make sure same size of clusters. Means K-means combine all data points that are near and merge them in one cluster. Our data we know does not follow normal distribution and has non-parametric nature. So based of data nature, different clustering should be used. Discretization using K-means is affecting the performance of stability metrics and producing strange results.

Table 5.9: Summary of the prediction analysis in the form of accuracy using Naive Bayes Classifier.

	Maintenance Effort Prediction	Future Maintenance Effort Prediction
All Stability Metric	67.984%	73.164%
PSM Metrics Suite	67.513%	78.543%
All Five Stability Metrics Other than PSM	69.942%	78.22%

From above analysis of prediction using Nave Bayes classifier, we can say that proposed metrics (PCS, IPIS and EPIS) are producing better results. So in our next analysis we use proposed metrics to predict maintenance effort and future maintenance effort using basic setup of six different classifier. Table 5.10 and figure 5.2 provide details of six classification techniques accuracy for prediction of maintenance effort and future maintenance effort. Results of those six classifier are given below:

- **Nave Bayes [59]:**For prediction of maintenance effort nave bayes produce 69.513% accuracy. Whereas for prediction of future maintenance effort it produce 78.543% accuracy.

- **Decision Tree [60]:**For prediction of maintenance effort decision tree produce 71.465% accuracy. Whereas for prediction of future maintenance effort it produce 81.203% accuracy.

- **Fuzzy Rules [61]:**For prediction of maintenance effort fuzzy rules produce 72.981% accuracy. Whereas for prediction of future maintenance effort it produce 81.59% accuracy.

- **Random Forest [62]:**For prediction of maintenance effort random produce 71.719% accuracy. Whereas for prediction of future maintenance effort it produce 83.51% accuracy.

- **Neural Network [63]:**For prediction of maintenance effort neural network produce 71.646% accuracy. Whereas for prediction of future maintenance effort it produce 84.095% accuracy.

- **Support Vector Machine (SVM)[64]:**For prediction of maintenance effort SVM produce 74.148% accuracy. Whereas for prediction of future maintenance effort it produce 84.268% accuracy.

So from above results we can easily say tha SVM out perform all other classifier in prediction of maintenance effort and future maintenance effort.

Table 5.10: Summary of the prediction analysis in the form of accuracy using Six Classifier.

	Maintenance Effort	Future Maintenance Effort
Nave Bayes	69.513	78.543
Decision Tree	71.465	81.203
Fuzzy Rules	72.981	81.59
Random Forest	71.719	83.51
Neural Network	71.646	84.095
SVM	74.148	84.268

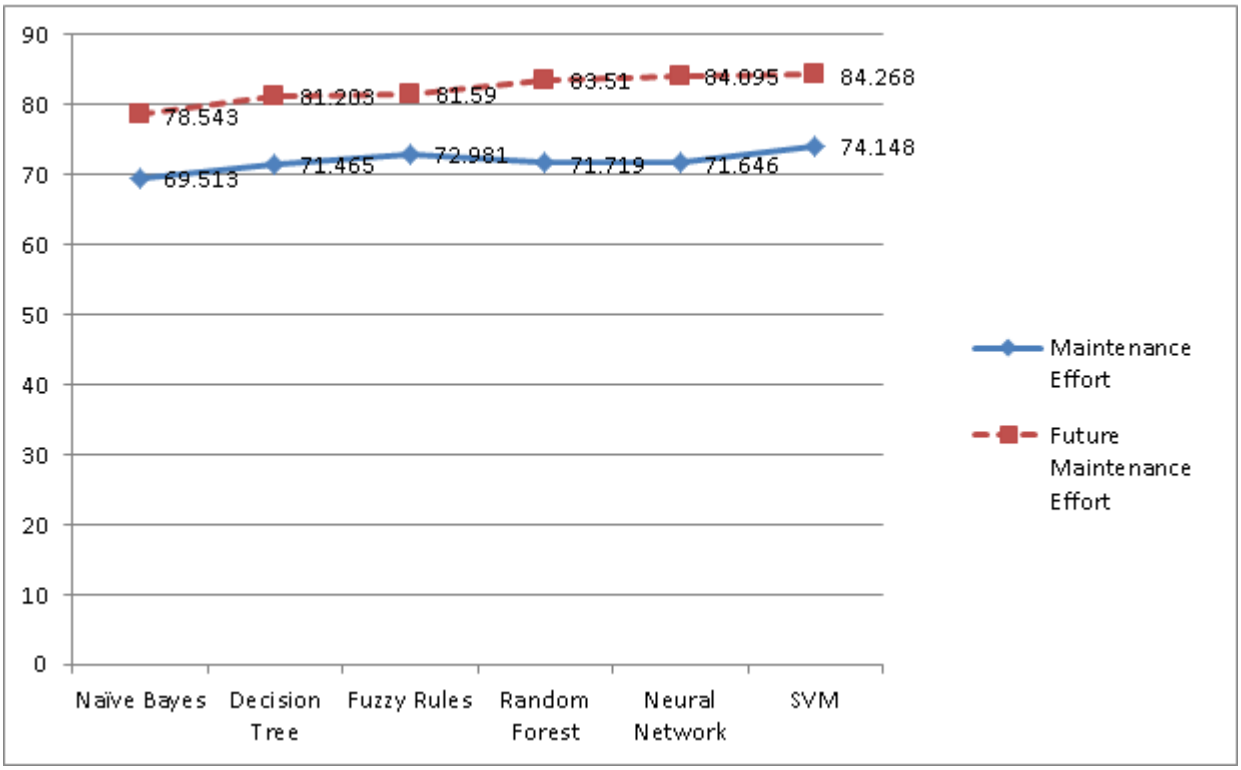


Figure 5.2: Prediction of maintenance effort and future maintenance effort using regression.

CHAPTER 6

CONCLUSION AND FUTURE WORK

6.1 Discussion

In this study, new PSM metric suite for package stability is proposed. It measures different stability aspects of a package. PCS calculates the change in the contents of a package. In the calculation, it measures the change in the contents of classes and interfaces. This metric mainly helps to understand the change in functionality of a package. IPIS calculates the change in the connections of same package contents (classes and interfaces). It can be used to judge the change in the cohesion of a package. Low IPIS means that a lot of package connections are change and testing of the package is required. EPIS measures the change in external connections of package contents (classes and interfaces) with contents (classes and interfaces) of other packages. It helps to understand the change in

overall architecture of the system and also the coupling of the package. Low EPIS means that a lot of external connections are removed, so testing of the package is required. PSM Metric suite is valid to measure according to Kitchenham et al. [1] framework. Kitchenham framework validates that it holds the representation condition and it can easily be used for stability comparison between packages.

We have used unchanged count, because our target was to calculate stability. Using our defined properties and formula instability of a package can also be calculated by counting changed properties instead of unchanged. In order to avoid calculation use below formula to calculate instability.

$$Instability = 1 - Stability$$

In empirical validation, correlation analysis with maintenance effort is performed. Maintenance effort counts line of codes that are modified, deleted or added in new version. Analysis shows that PSM metric suite has good negative correlation with maintenance effort. Results also shows that PCS has the best correlation with maintenance effort among all content based stabilities, while EPIS has the best correlation with maintenance effort among all interactions based stabilities. Hence we can say that PSM metric suite is a valid package stability metric suite.

We also performed correlation analysis with existing stability metrics. Purpose of this analysis was to identify relationship between PSM stability metric and six existing stability metrics. Martin's package instability metric uses dif-

ferent definition of stability. It just measure the dependency of other packages on a package using coupling measurements only and does not compare two versions of software. Analyses shows that only PSM metric suite has no correlation with Package stability. Our propose package content stability (PCS) has good correlation relationship with two content-based stabilities PCSM and PCII, while PCS has weak correlation with other two content-based stabilities PNomStab and PSDI. Reason is that PNomStab and PSDI consider changes only in number of methods and name of classes respectively.

Our proposed inter-package interaction based stability metric, EPIS has moderate correlation with another inter-package interaction based stability metric PASM. This relationship is not strong because PASM considers only association interaction between classes while EPIS considers interaction based on inheritance, aggregation, association, and dependency. EPIS also has weak correlation relation with PSDI and PNomStab. Reason for this relationship is that PNomStab considers changes in the number of methods, while PSDI considers changes in name of classes. Changes in methods and name of classes affect the inter-package interactions.

Our proposed intra-package interaction based stability metric, IPIS does not has strong or moderate correlation with any existing metric because no existing metric measure changes in intra-package interactions. Whereas IPIS also has weak correlation relation with PSDI and PNomStab. Reason for this relationship is that PNomStab considers changes in the number of methods, while PSDI considers

changes in name of classes. Changes in methods and name of classes affect the intra-package interactions.

Principal component analysis is used to determine that whether PSM metric suite cover new dimension and aspect of package stability or not. PCA results shows that PCS covers a new dimension in content-based stability of package. Whereas EPIS and IPIS also present new dimension of interaction-based stability metric. Finally prediction analysis of maintenance effort using linear regression shows that PSM metric suite improve the prediction accuracy. It also shows that from content point of view PCS best predict maintenance effort whereas from interaction point of view EPIS best predict maintenance effort. We also perform experiments to analyze the prediction performance of regression and classification techniques. From regression techniques we used linear regression, quadratic regression, polynomial regression with degree 3 and polynomial regression with degree 4. Polynomial regression with degree 4 produces best result as compared to other regression technique. From classification techniques we used Nave Bayes , decision tree, fuzzy rule, random forest, neural network and support vector machine. We found that SVM out perform other classification technique in prediction of maintenance effort and future maintenance effort.

6.2 Thread to Validity

6.2.1 Construct Validity

Construct validity means the experiment design decision can effect our results. Below are some of the construct validity threads.

- We ignored addition changes in proposed metrics, so we may have lost some useful information because of it.

- According to recommendation of different researchers and popularity of maintenance effort, we have used it as indicator of maintainability. This metric is validated by multiple researchers in different studies using different techniques. But still this is purely limited to change in line of code and different line of code may have different effect.

6.2.2 External Validity

Thread that are not in our control and can be caused by external factors. Below are some external validity threads.

- A large number of packages of software remains stable. This would have influence our analysis.

- Our experiments are limited to Java projects only. This can restrict our result and its implications to Java only.

- In our experiments we have used open source software. But we make sure

that they are widely used, famous, continuously evolving and belong to different domains.

- We have used Knime tool for classification and regression analysis which is very famous, easy to use and have a lot of good implemented techniques. Errors in Knime can effect our results.

6.3 Conclusion

Stable software architect reduces maintenance effort and cost. Packages are the intermediate level entities in object oriented design and help alot to make system architecture simple and understandable. So packages with good stability will increase the overall stability of system and reduce maintenance effort and cost. In our study, we have proposed metric suite to calculate package stability by different aspects. These aspects includes package contents, internal package connections and external package connections. Our metric suite cover more aspects and factors, so it will provide better identification of stability.

We have studied and proposed package stability metric for three aspects contents, internal package connections and external package connections. For calculation of package content stability we have studied and included eight class/interface properties/factors that can affect the stability of package. These factors represents structure of package elements so change in this structure will effect package stability. For calculations of package internal and external connections stability we have identified four type of possible relations between classes, two type of pos-

sible relations interfaces and four type of relations between classes and packages. These relations include inheritance, aggregation, composition, dependency and association. Change in these relations will effect the overall design of system and in result will effect the stability and maintenance.

We have validated our PSM metric suite theoretically using two different frameworks. For empirical validation of our PSM metric we have used five open source java software from diverse domain. We have found negative correlation of our metric with maintenance effort. We have also found positive correlation with existing package stability metric which are based on changes in line of code and class names.

6.4 Future Work

Future work of our research are mentioned below:

- Study can be conducted to investigate relationship of proposed stability metrics with other software characteristics like understandability, testability and software faults.
- Prediction of testability and software faults using proposed metrics can be done.
- We have assigned same weight to all properties, so study to find the affect of different weights can also be conducted.
- Study to define thresholds for proposed metric can be conducted.

REFERENCES

[1] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a framework for software measurement validation," *IEEE Transactions on Software Engineering*, vol. 21, no. 12, pp. 929–944, Dec 1995.

[2] Y. S. Hassan, *Measuring software architectural stability using retrospective analysis*. ProQuest, 2007.

[3] J. A. Dallal and L. C. Briand, "An object-oriented high-level design-based class cohesion metric," *Information and Software Technology*, vol. 52, no. 12, pp. 1346 – 1361, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910001552

[4] N. Fenton and J. Bieman, *Software metrics: a rigorous and practical approach*. CRC Press, 2014.

[5] M. Alshayeb, M. Naji, M. O. Elish, and J. Al-Ghamdi, "Towards measuring object-oriented class stability," *IET software*, vol. 5, no. 4, pp. 415–424, 2011.

[6] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *Journal of*

Systems and Software, vol. 82, no. 6, pp. 981 – 992, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121208002793

[7] M. Alshayeb, "On the relationship of class stability and maintainability," *IET Software*, vol. 7, no. 6, pp. 339–347, December 2013.

[8] M. E. Fayad and A. Altman, "Thinking objectively: An introduction to software stability," *Commun. ACM*, vol. 44, no. 9, pp. 95–, Sep. 2001. [Online]. Available: http://doi.acm.org/10.1145/383694.383713

[9] R. Martin, "Stability–c++ report," Tech. rep, Tech. Rep., 1997.

[10] N. L. Soong, "A program stability measure," in *Proceedings of the 1977 Annual Conference*, ser. ACM '77. New York, NY, USA: ACM, 1977, pp. 163–173. [Online]. Available: http://doi.acm.org/10.1145/800179.810197

[11] S. S. Yau and J. S. Collofello, "Some stability measures for software maintenance," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 6, pp. 545–552, Nov 1980.

[12] M. O. Elish and D. Rine, "Investigation of metrics for object-oriented design logical stability," in *Software Maintenance and Reengineering, 2003. Proceedings. Seventh European Conference on*, March 2003, pp. 193–200.

[13] M. Fayad, "Accomplishing software stability," *Commun. ACM*, vol. 45, no. 1, pp. 111–115, Jan. 2002. [Online]. Available: http://doi.acm.org/10.1145/502269.502308

[14] M. E. Fayad, "How to deal with software stability," *Commun. ACM*, vol. 45, no. 4, pp. 109–112, Apr. 2002. [Online]. Available: http://doi.acm.org/10.1145/505248.505278

[15] D. Grosser, H. A. Sahraoui, and P. Valtchev, "Predicting software stability using case-based reasoning," in *Automated Software Engineering, 2002. Proceedings. ASE 2002. 17th IEEE International Conference on*, 2002, pp. 295–298.

[16] W. Li, L. Etzkorn, C. Davis, and J. Talburt, "An empirical study of object-oriented system evolution," *Information and Software Technology*, vol. 42, no. 6, pp. 373 – 381, 2000. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584999000889

[17] M. Alshayeb and W. Li, "An empirical study of system design instability metric and design evolution in an agile software process," *Journal of Systems and Software*, vol. 74, no. 3, pp. 269 – 274, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412120400007X

[18] A. AbuHassan and M. Alshayeb, "A metrics suite for uml model stability," *Software & Systems Modeling*, pp. 1–27, 2016. [Online]. Available: http://dx.doi.org/10.1007/s10270-016-0573-6

[19] M. Ahmed, R. Rufai, J. AlGhamdi, and S. Khan, "Measuring architectural stability in object oriented software," *Stable Analysis Patterns: A True Problem Understanding with UML*, p. 21, 2004.

[20] L. Aversano, M. Molfetta, and M. Tortorella, "Evaluating architecture stability of software projects," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 417–424.

[21] S. A. Ebad and M. A. Ahmed, "Measuring stability of object-oriented software architectures," *IET Software*, vol. 9, no. 3, pp. 76–82, 2015.

[22] D. Rapu, S. Ducasse, T. Girba, and R. Marinescu, "Using history information to improve design flaws detection," in *Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings. Eighth European Conference on*, March 2004, pp. 223–232.

[23] D. Grosser, H. A. Sahraoui, and P. Valtchev, "An analogy-based approach for predicting design stability of java classes," in *Software Metrics Symposium, 2003. Proceedings. Ninth International*, Sept 2003, pp. 252–262.

[24] H. M. Olague, L. H. Etzkorn, W. Li, and G. Cox, "Assessing design instability in iterative (agile) object-oriented projects," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 18, no. 4, pp. 237–266, 2006. [Online]. Available: http://dx.doi.org/10.1002/smr.332

[25] Y. Zhao, Y. Yang, H. Lu, J. Liu, H. Leung, Y. Wu, Y. Zhou, and B. Xu, "Understanding the value of considering client usage context in package cohesion for fault-proneness prediction," *Automated Software Engineering*, vol. 24, no. 2, pp. 393–453, Jun 2017. [Online]. Available: https://doi.org/10.1007/s10515-016-0198-6

[26] A. Tripathi and D. S. Kushwaha, "A metric for package level coupling," *CSI Transactions on ICT*, vol. 2, no. 4, pp. 217–233, Jan 2015. [Online]. Available: https://doi.org/10.1007/s40012-015-0061-0

[27] R. C. Martin, *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2003.

[28] J. Garland, R. Anthony, and B. Lawrence, "Accomplishing software stability," in *Workshop on Accomplishing Software Stability OOPSLA*, 1999.

[29] 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, Jun 1994.

[30] S. Raemaekers, A. van Deursen, and J. Visser, "Measuring software library stability through historical version analysis," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, Sept 2012, pp. 378–387.

[31] M. Alshayeb, Y. Eisa, and M. A. Ahmed, "Object-oriented class stability prediction: A comparison between artificial neural network and support vector machine," *Arabian Journal for Science and Engineering*, vol. 39, no. 11, pp. 7865–7876, 2014. [Online]. Available: http://dx.doi.org/10.1007/s13369-014-1372-4

[32] K. Sethi, Y. Cai, S. Wong, A. Garcia, and C. Sant'Anna, "From retrospect to prospect: Assessing modularity and stability from software architecture,"

in *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, Sept 2009, pp. 269–272.

[33] J. Bansiya, "Evaluating framework architecture structural stability," *ACM Comput. Surv.*, vol. 32, no. 1es, Mar. 2000. [Online]. Available: http://doi.acm.org/10.1145/351936.351954

[34] M. Alenezi, "Software architecture quality measurement stability and understandability," *INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS*, vol. 7, no. 7, pp. 550–559, 2016.

[35] F. Handani and S. Rochimah, "Relationship between features volatility and software architecture design stability in object-oriented software: Preliminary analysis," in *2015 International Conference on Information Technology Systems and Innovation (ICITSI)*, Nov 2015, pp. 1–5.

[36] E. Constantinou and I. Stamelos, "Architectural stability and evolution measurement for software reuse," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, ser. SAC '15. New York, NY, USA: ACM, 2015, pp. 1580–1585. [Online]. Available: http://doi.acm.org/10.1145/2695664.2695895

[37] M. Alenezi and F. Khellah, "Evolution impact on architecture stability in open-source projects," *International Journal of Cloud Applications and Computing (IJCAC)*, vol. 5, no. 4, pp. 24–35, 2015.

[38] M. Mattsson and J. Bosch, "Stability assessment of evolving industrial object-oriented frameworks," *Journal of Software Maintenance: Research and Practice*, vol. 12, no. 2, pp. 79–102, 2000.

[39] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and Intelligent Laboratory Systems*, vol. 2, no. 1, pp. 37 – 52, 1987, proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0169743987800849

[40] H. Abdi and L. J. Williams, "Principal component analysis," *Wiley Interdisciplinary Reviews: Computational Statistics*, vol. 2, no. 4, pp. 433–459, 2010. [Online]. Available: http://dx.doi.org/10.1002/wics.101

[41] K. H. Zou, K. Tuncali, and S. G. Silverman, "Correlation and simple linear regression," *Radiology*, vol. 227, no. 3, pp. 617–628, 2003, pMID: 12773666.

[42] "Buddi - https://sourceforge.net/projects/buddi/." [Online]. Available: https://sourceforge.net/projects/buddi/

[43] "Jhotdraw - https://sourceforge.net/projects/jhotdraw/." [Online]. Available: https://sourceforge.net/projects/jhotdraw/

[44] "Kolmafia - https://sourceforge.net/projects/kolmafia/." [Online]. Available: https://sourceforge.net/projects/kolmafia/

[45] "Talend - https://www.talend.com/." [Online]. Available: https://www.talend.com/

[46] "Unitime - http://www.unitime.org/." [Online]. Available: http://www.unitime.org/

[47] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111 – 122, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/016412129390077B

[48] M. R. Berthold, N. Cebron, F. Dill, T. R. Gabriel, T. Kötter, T. Meinl, P. Ohl, K. Thiel, and B. Wiswedel, "Knime - the konstanz information miner: Version 2.0 and beyond," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, pp. 26–31, Nov. 2009. [Online]. Available: http://doi.acm.org/10.1145/1656274.1656280

[49] "Javaparser - https://github.com/javaparser/javaparser." [Online]. Available: https://github.com/javaparser/javaparser

[50] J. Hauke and T. Kossowski, "Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data," *Quaestiones Geographicae*, vol. 30, no. 2, p. 87, 06 2011, copyright - Copyright Versita Jun 2011; Last updated - 2016-10-08. [Online]. Available: https://search.proquest.com/docview/1323984192?accountid=27795

[51] V. Klema and A. Laub, "The singular value decomposition: Its computation and some applications," *IEEE Transactions on Automatic Control*, vol. 25, no. 2, pp. 164–176, Apr 1980.

[52] A. B. Nassif, D. Ho, and L. F. Capretz, "Towards an early software estimation using log-linear regression and a multilayer perceptron model," *Journal of Systems and Software*, vol. 86, no. 1, pp. 144 – 160, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121212002221

[53] S. Almugrin, W. Albattah, and A. Melton, "Using indirect coupling metrics to predict package maintainability and testability," *Journal of Systems and Software*, vol. 121, pp. 298 – 310, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016412121600056X

[54] G. G. S. G. B. L. . M. M. Vernazza, T., "Defining metrics for software components." *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics.* [Online]. Available: http://www.jsoftware.us/vol11/166-CS006.pdf

[55] X. Yan and X. G. Su, *Linear Regression Analysis: Theory and Computing.* River Edge, NJ, USA: World Scientific Publishing Co., Inc., 2009.

[56] S. Chatterjee and A. S. Hadi, "Influential observations, high leverage points, and outliers in linear regression," *Statistical Science*, vol. 1, no. 3, pp. 379–393, 1986. [Online]. Available: http://www.jstor.org/stable/2245477

[57] R. M. Heiberger and E. Neuwirth, *Polynomial Regression.* New York, NY: Springer New York, 2009.

[58] A. Ahmad and L. Dey, "A k-mean clustering algorithm for mixed numeric and categorical data," *Data and Knowledge Engineering*, vol. 63, no. 2, pp. 503 – 527, 2007.

[59] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine Learning*, vol. 29, no. 2, pp. 131–163, Nov 1997. [Online]. Available: https://doi.org/10.1023/A:1007465528199

[60] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 3, pp. 660–674, May 1991.

[61] P. P. Angelov and X. Zhou, "Evolving fuzzy-rule-based classifiers from data streams," *IEEE Transactions on Fuzzy Systems*, vol. 16, no. 6, pp. 1462–1475, Dec 2008.

[62] V. Svetnik, A. Liaw, C. Tong, J. C. Culberson, R. P. Sheridan, and B. P. Feuston, "Random forest: a classification and regression tool for compound classification and qsar modeling," *Journal of Chemical Information and Computer Sciences*, vol. 43, no. 6, pp. 1947–1958, 2003, pMID: 14632445.

[63] H. A. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, Jan 1998.

[64] J. Suykens and J. Vandewalle, "Least squares support vector machine classifiers," *Neural Processing Letters*, vol. 9, no. 3, pp. 293–300, Jun 1999. [Online]. Available: https://doi.org/10.1023/A:1018628609742

Vitae

- Name: Jawad Javed Akbar Baig
- Nationality: Pakistani
- Date of Birth: 25/07/1990
- Email: *jawadjavedbaig@gmail.com*
- Permenant Address: 108 Atta Turk Block, Garden Town, Lahore, Pakistan
- Academic Background: Jawad Javed Akbar Baig completed his Bachelor degree in Software Engineering from University of the Punjab in July 2012. He worked in industry as Software Engineer at Techlogix for two year and as Senior ERP Consultant at Confiz for one year. In August 2015, he joined King Fahd University of Petroleum and Minerals.