SOURCE CODE SIMILARITY ANALYSIS TO ENHANCE THE ACCURACY OF SOFTWARE DEFECT PREDICTION MODEL

by

Md. Rayhanul Islam
Registration No: H-562
Session: 2008-2009

A Thesis
Submitted in partial
fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

Institute of Information Technology
University of Dhaka
DHAKA, BANGLADESH

© Md. Rayhanul Islam, 2015

SOURCE CODE SIMILARITY ANALYSIS TO ENHANCE THE ACCURACY OF SOFTWARE DEFECT PREDICTION MODEL

Md. Rayhanul Islam

Approved:

Signature *Date*

Supervisor: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Zerin Begum

Committee Member: Dr. Kazi Muheymin-Us-Sakib

Committee Member: Dr. Muhammad Mahbub Alam

Committee Chair: Shah Mostafa Khaled

To my parents and the people who are continuously working to make this planet peaceful.

Abstract

Software defect prediction is a process where software code metrics and past information are analyzed to predict defects. The early estimation of software defects helps software industries to develop software with minimum cost and effort. The accuracy and performance of a prediction model depends on the Dataset used to train the model. To provide proper Dataset in the training process, clustering algorithms can be applied to form clusters with similarities that contains less variablities among the Dataset. Due to multidimensionalities in the software engineering Dataset, it has been found that variability reducing clustering algorithms cannot perform well until multidimensionalities are reduced.

In this thesis, the Dataset similarities are analyzed from two perspectives - first from the source code, and then from the code metrics points of views. The similarity of software source code is analyzed by using a new algorithm called the Package Based Clustering (PBC) and the similarity of code metrics is analyzed using Similarity Analysis by Reducing Code Metrics' dimension (SARCM) technique.

PBC groups the similar and related classes from an Object Oriented (OO) system. For that purpose, it extracts the package information from the source code and classifies all OO files according to package information. Then it considers each package as a cluster and also merges small clusters so that the prediction model can work.

In SARCM, it reduces the dimensions of the software engineering Dataset into two latent variables based on the significance of code metrics to software defects. It applies regression analysis on the available Dataset to find the significance of code metrics. Then it merges code metrics based on their positive or negative significance. Now the distance based clustering algorithm (such as DBSCAN) can group those Dataset into multiple clusters based on their similarities.

Experiments have been performed on some prominent open source software Dataset. Results show that training the defect prediction model by similar Dataset improves the

accuracy and performance of the prediction model. The experiments proved that, the proposed PBC outperforms the prediction model using the entire system and BorderFlow clustering approach in terms of accuracy. In another technique, results show that the SDP model considering SARCM based DBSCAN and WHERE clustering techniques perform better than the PCA based DBSCAN and WHERE clustering techniques.

Acknowledgments

I take this opportunity to express my profound gratitude and deep regards to my thesis Supervisor Dr. Kazi Muheymin-Us-Sakib, Director and Associate Professor, Institute of Information Technology, University of Dhaka, for his exemplary guidance, monitoring and constant encouragement throughout the course of thesis. The blesses, helps and guidances, given by him time to time shall carry me a long way in the journey of my life on which I am about to embark.

I would like to convey my heartfelt gratitude to all faculty members, Institute of Information Technology, University of Dhaka, for their support, inspiration, criticism and constructive feedback which has immensely strengthened my confidence during my thesis.

I also take this opportunity to express a deep sense of gratitude to DSSE Student Research group, Institute of Information Technology, University of Dhaka, specially Asif Imran, Lecturer, IIT and Nadia Nahar (MSSE0301) for their cordial support, valuable information and guidance, which helped me in completing this task through various stages.

I am expressing ever gratefulness to all my fellow classmates whose advice, feedback and cooperation is truly incomparable.

I am also thankful to Ministry of Information and Communication Technology, Government of the People's Republic of Bangladesh for granting me ICT fellowship of $2014-15$ 1st round.

Finally, I am indebted to my parents for the many years of hard work and sacrifices they have made to support me. Without them, this thesis would never have been started.

Publications

The motivation to proceed with the research and achieve completion got a tremendous boost when our research paper got accepted in reputed conferences. The publication made during the course of this research has been listed below.

1. R. Islam and K. Sakib. A package based clustering for enhancing software defect prediction accuracy. In 17th International Conference on Computer and Information Technology (ICCIT), 2014, pages 81−86, Dec 2014.

Table of Contents

List of Tables

List of Figures

List of Java Source Codes

Chapter 1

Introduction

In this chapter, the motivation of doing research on software defect prediction is illustrated, by which the research questions are invoked. To address those questions, the whole research is carried out, and the short description of research methodology and outcomes are outlined to answer these research questions.

1.1 Introduction

Software defect prediction analyzes source code, code metrics, code churn and past defect information by applying machine learning or statistical models to predict defects for the future releases of software. It enables to assess the software quality as well as the source code before its final release to the end user. The US National Institute of Standards and Technology (NIST) estimated that software industries spend $59.5 billion for identifying software defects and failures per year [1]. To provide the quality full software to end user, every software company spends more than 40% of the total budget of software for testing purpose [2]. As a result, for ensuring the quality and estimating the complexity, software defect prediction model has been emerged to solve common problems that software industries face to maintain software quality.

1.2 Motivation

Knowing the possible causes of software defects early, could help on planning, controlling and executing software development activities because it assists in decision making of Project Managers (PM). It helps to develop software with minimum cost and effort. Soft-

ware project with many defects lack the software quality and increases the development cost. Using Software Defect Prediction (SDP) model, one can able to -

- Identify potential defect prone software.
- Predict number of defects.
- Identify possible causes of defects.

The SDP model usually takes the past defect information and code metrics as input and establishes relationship among those to predict defects. However, those input Dataset contains lots of variabilities such as heterogeneity among code metrics, which hinder the learning procedure of a SDP model. The success of defect prediction approaches depends on how accurately those could minimize the variablilities among the Dataset.

Grouping the Dataset into multiple clusters is the prominent solution to minimize the Dataset variabilities. There are number of clustering algorithms such as BorderFlow [3], Subtractive Clustering [4], WHERE clustering [5, 6], etc. which are common in SDP techniques. These clustering techniques use dependencies among code metrics, class coupling, etc. to group the software engineering Dataset.

Besides varaibilities, those Dataset are also multidimensional because each entry contains lots of properties such as coupling between objects, number of public methods, etc. There are few defect prediction approaches, which use Principal Component Analysis (PCA) before applying clustering techniques to reduce the dimension of Dataset [5, 6]. However, considering minimum number of components raise the question that, can a subset of components justify the impact of the whole Dataset.

Distance based clustering algroithms need to plot the Dataset on the two dimensional plane to measure the distance among the objects. If multidimensionality among the Dataset is properly reduced, the clustering algorithms may group the Dataset in a better way with minimum variabilities which may improve the learning procedure of the SDP model. Thus

reducing varaibilities through minimizing multidimensionality may improve the accuracy and performance of the SDP model.

1.3 Research Question

Keeping the research indications in view, it has been realized that there exists enough scope to improve the software defect prediction accuracy. In this research the objectives are confined to the followings.

How to reduce the multidimensionalities of the Dataset to be fitted to the variability minimizing clustering approaches using data similarity analysis. More specifically,

1. How to identify the class level similarities to cluster the related and similar classes for the purpose of predicting defects in Java using package information? A package is a collection of related and similar classes. It acts as a component if the software system is built properly. So, OO classes within a package are related to each other than the classes outside the package. If a software is divided into multiple clusters based on the package information, the prediction model will get similar and related Dataset for training purpose. As a result, the accuracy of the prediction model should be improved.

2. How to reduce the code level data multidimensionality using the code metrics' impact to defects? Usually, software engineering Dataset are multidimensional because each entry contains lots of code metrics attributes such as coupling between objects, lack of cohesion in methods, number of public methods, etc. Due to this multidimensionalities, clustering algorithms cannot perform well. So, if it is possible to minimize the dimension of the Dataset based on the significance of code metrics impact to defects, it will definitely help the defect prediction model.

3. How to fit the dimension reduced Dataset to distance based clustering algorithms to improve the performance of the SDP? The dimension reduced Dataset contains only two dimensions where similar entries get closer values. If the Dataset is plotted on the two dimensional plane, the similar objects reside to each other than disimilar objects. Any distance based clustering algorithm can now group the Dataset based on their similarities.

1.4 Contribution

In answering these above research questions, this thesis contributes to reduce variablities by reducing multidimensionalities to improve the accuracy of the defect prediction model for object oriented software built by Java. These contributions are summarized as follows.

Firstly, this thesis proposes Package Based Clustering (PBC) to group related and similar object oriented classes that form packages in Java programing convention. It groups software based on the package information so that the similar and related classes belong to one cluster. The experimental results show that the software defect prediction model considering PBC is better than considering the entire system or BorderFlow algorithm [3].

Secondly, this thesis proposes a technique named as Similarity Analysis by Reducing Code Metrics (SARCM) to reduce the dimensions of software engineering Dataset based on the significance of code metrics to the number of software defects. It reduces the dimension in such a way where the similar objects of an OO system get closer values in the dimension reduced Dataset. This technique provides an environment where different distance based clustering algorithms can perform on the dimension reduced Dataset by plotting it in the two dimensional plane.

Thirdly, this thesis uses DBSCAN for the first time in the dimension reduced Dataset by SARCM to minimize variablities. Since the DBSCAN finds clusters based on the rech-

ability distance among different objects, clusters are formed based on the similarity among objects, because the dimension reduced Dataset contains closer value for similar objects. As a result, the SDP model gets proper training Dataset to improve the performance and accuracy of the model. Results show that the prediction model using SARCM based DBSCAN outperforms the prediction model considering PCA based DBSCAN.

1.5 Thesis Organization

The rest of this thesis is organized as follows.

1. **Chapter 2: Background Study** In this chapter, overview of software defect, defect prediction model, its classification, software code metrics and its classification will be discussed.

2. **Chapter 3: Package Based Clustering** In this chapter, Package Based Clustering (PBC) for grouping the source for the software defect prediction model will be presented.

3. **Chapter 4: Code Metrics Similarity Analysis using Dimension Reduction technique for SDP** In this chapter, The significance of code metrics to software defects will be analyzed to group the source code into multiple clusters to train the SDP model properly.

4. **Chapter 5: Discussion and Conclusion** In this chapter, retrospect on overall proposed approach for improving the accuracy of the software defect prediction model will be focused along with future research direction.

Chapter 2

Background Study

This chapter discusses briefly about the software defect prediction models along with various ways of predicting software defects. It also outlines the linear regression as the selected software defect prediction model and the most significant code metrics for OO system.

2.1 Introduction

Software Defect Prediction (SDP) is a process of predicting defects by using the code metrics and bug history of a targeted software to ensure the quality. The Software Quality Assurance (SQA) is a set of activities that ensures software system to meet a specific quality level. Software companies always concern to explore the defects of software before its release. To predict software defects, many researches have already been conducted for building prediction models (for example, Linear regression [7, 8], Logistic regression [9], Support Vector Machines (SVM) [10], Bayesian network [11], etc.). These models are popularly known as SDP models.

Usually, the SDP model is selected from the statistical or machine learning models to apply on the software engineering Dataset such as Promise Repository [12] to predict defects of software. Finally, the accuracy and performance of the prediction model is evaluated by using precision, recall, F-measure, coefficient of determination, etc [13]. The general overview of a SDP model is illustrated in Figure 2.1.

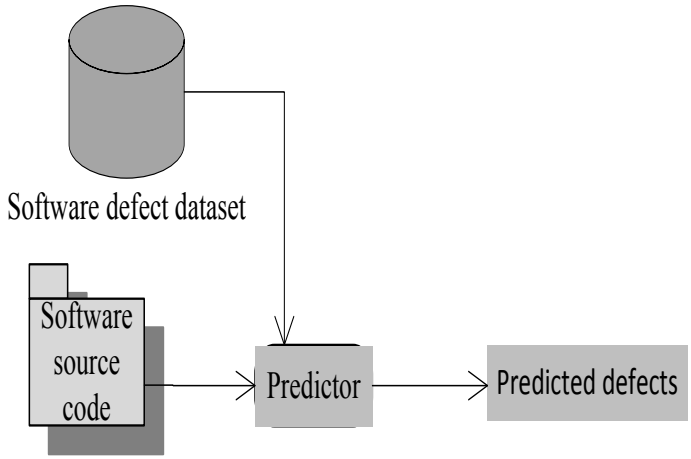

Figure 2.1: The general overview of a SDP model

2.2 Defination of Software Defect and SDP

The software defect is any flaws or errors in a software that produces unexpected results. A software programmer, designer or others, associated to the development of a software can make mistakes or errors while designing or building the software. These mistakes or errors are called defects [14, 15].

The term defect may vary based on the perception of the software developers, QA teams or Project Managers (PM). For example, bug databases of open source software contain user requests which are treated as bug correction requests by the user and often considered as feature requests by the developers of those systems. That means, what one stakeholder considers as a defect may not be perceived as the same by other stakeholders. In essence, software defect is any kind of deviation from stakeholders' expectation.

A SDP is a method that helps the software practitioners by predicting possible defects that the end product may contain. The software defect prediction model, also known as defect predictors use the software code metrics and the past defect information to predict defects for the current project. According to Brooks [16], half of the cost of software development is in unit and systems testing. Harold and Tahat also address that testing phase requires approximately 50% of the whole project schedule [17, 18]. The defect predictor

model is applied to the software project before testing to identify where the defects might exist. This allows PMs to efficiently allocate their limited resources.

The SDP model, though works well in predicting the software defect, there exists critical steps and decisions such as prediction model selection, code metrics selection, etc. that are needed to be considered when building the defect prediction model. The various ways of defect prediction and challenges that should be in account when predicting defects are described in the following subsections.

2.2.1 Selected Defect Prediction Model

The SDP uses a variety of modeling techniques to predict the defects of software. These prediction models can be categorized to the statistical models and machine learning algorithms. The statistical models formalize the relationship between the independent variables and the dependent variable to predict the next dependent variable. On the other hand, machine learning algorithms learn from the experiences that it gained. The selected defect prediction model considered in this thesis is described below.

Linear Regression Analysis

The regression analysis is a statistical process for estimating the relationships among the variables. It helps to understand how the values of the dependent variables change when the values of independent variables are changed. The performance of the regression analysis depends on the data used to train the regression model and how it relates to the regression model being used. There are many techniques for carrying out the regression analysis such as linear regression, logistic regression, step wise regression, nonparametric regression, etc [8]. In this thesis, the linear regression analysis has been chosen as prediction model

because there exists linear relationship among the variables of the used Dataset. The detail description of linear regression analysis is given below.

Linear regression analysis is the process of analyzing the relationships between variables, usually the variables are divided into dependent and independent variables. Let Y denote the dependent variable which will be predicted and $X_1, \therefore, X_k$ denote the independent variables. The linear regression analysis predicts the value of dependent variable (Y) by analyzing the independent variables ($X_1, \therefore, X_k$). The equation for computing the predicted value of Y is given in Equation 2.1.

$$Y = b_1X_1 + b_2X_2 + b_3X_3 + ... + b_nX_n + c \tag{2.1}$$

This formula is a straight-line function of each of the X variables, holding the others fixed, and the contributions of different X variables to the predictions are additive. The slopes of their individual straight-line relationships with Y are the constants b_1, b_2,..., b_k, the so-called coefficients of the variables. The coefficient (b_i) is the change in the predicted value of Y per unit of change in X_i. The additional constant c, the so-called intercept, is the prediction that the model would make if all the Xs are zero.

The linear regression fits in the Dataset when the variables in the Dataset have linear relationship among these. So, before selecting the linear regression as the prediction model, it is needed to find the relationship among the variables to decide whether linear regression is the best choice or not. There is also another analysis to determine the prediction model using residual values. If the residual values, generated from the regression analysis, are randomly distributed compared to two consecutive observations, in that case the linear regression analysis is perfect choice as a prediction model.

2.2.2 Other Defect Prediction Models

There exists lots of perdition models in the literature such as logistic regression [9, 19], Naive Bayes [20], Support Vector Machine (SVM) [10, 21, 22], etc. The short descriptions of those prediction models are given below.

Naive Bayes Classifier

It is the simple probabilistic classifier, based on Bayes theorem [11]. The classifier calculates a future probability for a class using the prior probability of that class. Given a set of objects of an OO system $x_1, x_2, x_3..., x_n$ and $c_1, c_2, c_3..., c_n$ are the corresponding defects of those objects. The conditional probability of c_n can be computed using Bayesian theorem using Equation 2.2.

$$P(C_k|x) = \frac{P(C_k)P(x|C_k)}{P(x)} \tag{2.2}$$

Support Vector Machine

Support Vector Machine (SVM) is a supervised learning model that analyzes data and recognizes patterns, especially designed for binary classification [10, 21, 22]. SVM utilizes mathematical programming to directly model the decision boundary between classes, for example, defective and non-defective. An SVM tries to find the maximum margin hyperplane, a linear decision boundary with the maximum margin between it and the training examples in defective class and the training examples in non-defective class. Therefore, the optimal separating hyperplane maximizes the margin of the training data.

Given a set of training objects each of which belong to one of two categories, an SVM training algorithm builds a software defect prediction model that assigns new examples into

one of the two categories (for example, defective and non-defective classes). The operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. This distance receives the important name of margin within SVMs theory.

K-Nearest Neighbor

K-Nearest Neighbor (KNN) is a type of instance based learning for classifying objects based on closest training examples in the feature space [23–25]. The training examples are mapped into multidimensional feature space which is partitioned into regions by class labels. An object is assigned to the class if it is the most frequent class label among the k nearest training samples. An Euclidean distance is used to compute the closeness to the samples. The number of neighbors can be set as a parameter considering the mean squared error for the training set.

Besides those discussed prediction models, other prediction models such as Bayesian network [11], Step Wise Linear Regression [26], Markov Chain [27], Artificial Neural Networks [28], etc. are widely used as prediction model in software defect prediction.

2.2.3 Challenges in Software Defect Prediction

Most of the SDP models use statistical and machine learning models for predicting software defects. The widely used SDP models in literature usually use public Dataset and sometimes use private Dataset to identify the relationship between software defects and code metrics. The success of a SDP model depends on various factors such as prediction model selection, Dataset selection, etc. to predict more accurately. Some challenges that are needed to be considered in software defect prediction are outlined below.

Dataset Collection

Every SDP model uses software Dataset such as software code mertics, defect information, etc. from the public data sources such as Promise Repository [12] and NASA Repository [29]. These software data sources contain lots of open source software code metrics and past defect information to help software engineering researchers. The SDP model using private Dataset can not be compared to other models because those cannot be reached. To make any SDP model acceptable to other, it is always appreciated to use public data source for the defect prediction.

Code Metrics Selection

Once the Dataset has been selected for the SDP model, it is now time to identify the most significant code metrics so that the prediction model can accurately relate the code metrics to software defects. There have been accomplished lots of researches to identify these code metrics. After reviewing these experiments, in this thesis, some code metrics have been considered for predicting defects such as Lines of Codes, Coupling between Objects, etc. (see Section 2.3 for detail description).

Model Selection

As mentioned above, there are lots of prediction models in the literature for SDP. Some of those models are already outlined in Section 2.2.1. The model selection process for defect prediction is very critical because its success depends on the selected model used for the prediction. Prior researches show that most of the SDP models use machine learning or statistical inferences such as Decision Tree, Linear Regression, Bayesian network, Support Vector Machine, Logistic Regression, etc [13]. Before selecting a prediction model, it

should be ensured that the prediction model must fit to the data and can explain the data accurately. In this thesis, Linear Regression has been chosen as the prediction model because the Dataset used in this experiment are tightly coupled to each other and residual values are randomly distributed for this prediction model.

Data Analysis

The success of software defect prediction process depends on the training process of the SDP model. Normally, the SDP model uses software code metrics and defect history as the training data. The software engineering data always contains some variablities which mislead the training process [30]. To provide the best data in the training process, many researches have been performed to find out the best set of data. Some researches suggest to group the Dataset by analyzing inter relationships, so that the SDP model gets proper training Dataset. So, before applying SDP model to the Dataset, it is needed to ensure that the prediction model fits to the data.

2.3 Software Code Metrics

The software metric or code metric is a quantitative measure of a software system [31]. There are lots of software metrics, for example, Method level, Class level, Component level, Process level, Cyclomatic complexity, etc [13]. These code metrics have been used for software defect prediction to improve the quality. The software code metrics have multidimensional usage in software industry such as defect prediction, schedule and budget planning, cost estimation, quality testing, software debugging, software performance optimization, etc. The well known OO and size metrics which have been used in this thesis for software defect prediction are reported below.

2.3.1 Object Oriented Design Metrics

When the object oriented software development paradigm has become popular, a lot of researches have been conducted to find the most important code metrics for the OO designed system. One of the most important OO code metrics, CK metrics, was proposed by Chidamber and Kemerer [31]. In addition to CK metrics, there are also other metrics suite such as MOOD [32], QMOOD [33], L&K [34], etc. Among all the OO design metrics, the CK metrics is the most influential code metrics and widely used to measure the quality of OO designed system [13, 35, 36]. The detail description of the CK metrics are given below.

Weighted Methods per Class (WMC) The WMC is the sum of all methods that belong to a class [31]. It indicates how much effort is required to develop and maintain a particular class. The low value of WMC points to greater polymorphism and the high value indicates the class is complex, difficult to maintain and reuse. The classes with high value of WMC are often re-factored into two or more classes to make the source code simple, maintainable and reusable.

Depth of Inheritance Tree (DIT) The DIT simply counts the number of ancestors of a class [31]. The high value of DIT means the class inherits greater number of methods which makes a class complicated and complex. To the Object Oriented Programing (OOP) point of view, a class should maintain the single responsibility principle. When a class inherits a lot of classes, it is most likely that the class inherits a lot of methods which is the violation of the OOP principle. As a result, the designed system cannot utilize the benefits of OOP principles.

From the Figure 2.2, the DIT values of different classes are given below:

- DIT (C_0) = 0
- DIT (C_1) = 0
- DIT (C_2) = 1

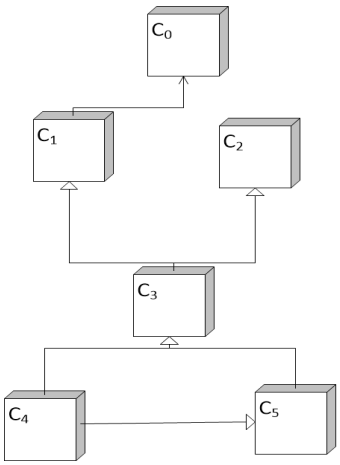

Figure 2.2: The class hierarchy of an OO system

- DIT (C_3) = 2
- DIT (C_4) = 3
- DIT (C_5) = 4

Number of Children (NOC) The NOC simply counts the immediate sub-classes of a given class [31]. It is also a measure of how many sub-classes are going to be inherited by the methods of a particular parent class. From Figure 2.2, the NOC values of different classes are given below:

- NOC (C_0) = 1
- NOC (C_1) = 1
- NOC (C_2) = 1
- NOC (C_3) = 2
- NOC (C_4) = 3
- NOC (C_5) = 4

The NOC value also gives the potential influence of a class on the system designing. The high value of NOC ensures the high usage of the inheritance and the greater possibility of the reusability in a system. Sometimes, it may also be a case of misuse of sub-classing and increase the system complexity that results more testing of the methods to ensure the quality of a software.

Response for a Class (RFC) The RFC metric is the total number of all methods that can be executed in response to a message received by a class [31]. It is the sum of the methods of a class and all distinct methods that are invoked directly within the class methods. If a class invokes large number of methods in a response, the testing and debugging of the class becomes more complicated and the tester requires a greater level of understanding to test the method.

Coupling Between Objects (CBO) The CBO simply counts the number classes to which one single class is connected [31]. A class can be coupled to other classes through parameter passing, shared variables, common database access or direct method calling, etc. From the example in Figure 2.3, the coupling value of these classes are listed below:

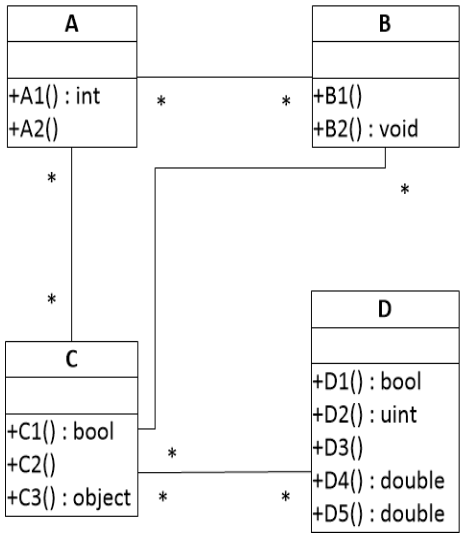

Figure 2.3: The coupling among different classes in an OO system

- CBO(A)=2

- CBO(B)=2
- CBO(C)=3
- CBO(D)=1

The High value of CBO is always detrimental to modular design and prevents reuse of classes. When a class has high CBO value, it means that the class has high dependency to the other classes. So, the corresponding class has high possibility of having defects than others and needs more testing compared to other. On the contrary, the low value of CBO indicates the simplicity of a class and it is easy to reuse in another application.

Lack of Cohesion of Method (LCOM) The cohesion measures how well the methods in a class are connected to each other [31]. It actually means the number of methods that do not share common fields like methods, properties, etc. A class is called cohesive class if it performs only one function. So, the lack of cohesion means that the class performs more than one responsibility. From the OOP point of view, a class should perform only one responsibility. The LCOM value is used to determine whether the class violates the single responsibility principle or not.

LCOM = 1 means that the class is cohesive class and LCOM $\geq$ 2 means that the class has more than one responsibility and should be split to make the design simple. High cohesion among the methods is always desirable as it helps to achieve encapsulation. Low cohesion indicates the inappropriate design and high complexity which leads to create fault prone software.

In Figure 2.4, a class consists of methods A through E and variables x and y. A calls B and B accesses x. Both C and D access y. D calls E, but E does not access any variables. This class has 2 unrelated components. The class can be split into the following components such as A, B, x and C, D, E, y. So, it will be appropriate to split the class into two classes. For this class, the LCOM value is 2.

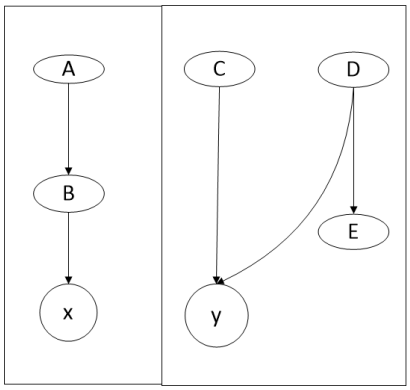

Figure 2.4: A simple overview of lack of cohesion in a class

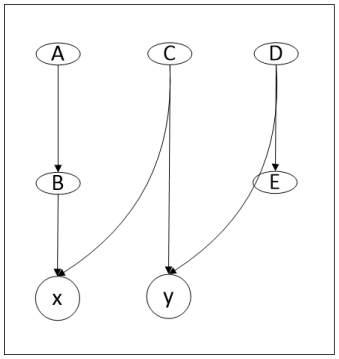

Figure 2.5: A simple overview of a cohesive class

In Figure 2.5, Method C access x to increase cohesion. Now the class consists of a single component where each and every variable and method are connected to each other. So the LCOM value for this class is 1. It is a cohesive class.

2.3.2 Software Size Metrics

Besides the CK metrics [31], the well known software size metrics which are the Number of Public Methods (NPM) [3] and the Lines of Code (LOC) [3] have been employed for defect prediction. The short description of those are given below.

Number of Public Method (NPM) It is the sum of public methods in a class that can accessible from outside of the class [3]. The high value of NPM indicates that the

class is highly coupled with other classes because these are interfaces by which classes communicates to each other. The high value of NPM also shows that the corresponding class is too complex and has many responsibilities which complicates the system.

Lines of Code (LOC) The LOC simply counts the total number of lines from the source code by avoiding pure whitespace and lines containing only comments [3]. It measures the volume of the source code. It can only be used to compare projects, built by the same programing language and the same coding standards. It is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or maintainability once the software is produced.

2.3.3 Other Code Metrics

Besides the above discussed software code metrics, there are also other code metrics such as Halstead Metrics [37], McCabe metrics [38], etc. All of these code metrics are frequently used for defect prediction. The McCabe metrics is used to capture the structural complexity of the source codes [38]. It includes cyclomatic complexity v(g), design complexity ev(g) and essential complexity iv(g). Cyclomatic complexity measures the number of linearly independent paths to determine the source codes complexity. The Halstead Code Metrics [37] is another way to measure the complexity of a program's source code. It measures the complexity and difficulty of the source code using the number of distinct operators (n_1), the number of distinct operands (n_2), the total number of operators (N_1) and the total number of operands (N_2). It can be used to calculate the program's length, difficulty, effort, bug, volume, etc.

2.4 Conclusion

This chapter discusses briefly about the software defect prediction models along with various ways of predicting software defects. It also highlights the linear regression as the selected software defect prediction model with the challenges that must be taken into account when predicting software defects. Lastly, It lists out the most significant code metrics, that is CK metrics for OO system. The next chapter discusses about the proposed Package Based Clustering (PBC) for software defect prediction.

Chapter 3

Package Based Clustering

In this chapter, the proposed Package Based Clustering, selected Dataset, prediction model and its validation process are outlined. Finally, the result analysis is carried to show the importance and effectiveness of the proposed technique.

3.1 Introduction

Software defect prediction models use software code metrics and knowledge from previous projects to predict software defects for future releases. Early estimating the faultiness of a software can help practitioners to assess their current project status as well as it reduces the software development cost. To predict defects before software testing process, many researches have been conducted using different defect prediction models, for example, Neural network, Naive Bayes, Regression modeling, Decision tree, etc. [13]. Most of those models predict software defects by considering the entire system. Due to variability in software data, prediction models considering the entire system do not always provide the desired results [30]. So, if software defects are predicted by partitioning the software into multiple clusters, it may produce better results than those [3].

Software clustering can be accomplished by using different clustering algorithms such as BorderFlow [3], subtractive clustering [4], etc. Those clustering algorithms use software code metrics, for example, Class level, Method level, Process level, etc. or source code dependencies such as class reference to form clusters from the software. Although, clustering algorithms help defect prediction models to improve the accuracy and performance, all of those cannot always provide the best results because of the inefficiency of clustering algorithms to group the software.

In recent years, several software defect prediction models have been developed to predict software defects. Menzies et al. [5, 6] proposed that learning from software clusters with similar characteristics is better than learning from the entire system, because it may falsify the data used by the prediction model. It performs clustering by using WHERE clustering technique which only considers the code metrics and learning treatment from pairs of neighboring clusters. Scaniello et al. [3] proposed a defect prediction model using clustering where it considers step wise linear regression model to predict defects. It uses BorderFlow algorithm to form clusters among the related classes by using references between methods and attributes. Sidhu et al. [4] proposed a software fault prediction model which uses subtractive clustering algorithm and fuzzy inference system for early detection of faults. To predict the faults, Zimmermann et al. [39] proposed the use of components for grouping software metrics, because the likelihood to fail of a component is dependent on its problem domain.

In this dissertation, a new way to group the source code for software defect prediction is proposed to improve the accuracy by using Package Based Clustering (PBC) rather than the entire system. PBC groups the software, implemented using Java, into a number of clusters using package information of each OO classes. To find clusters, PBC lists out all OO classes by using textual analysis of source code. It then reads those classes to extract the package information to form clusters. If the number of OO classes of a cluster is smaller than the number of explanatory variables used in the defect prediction model, it combines small clusters to enable those in order to apply prediction model.

To validate the PBC, an experiment has been conducted on open source software which are Ant, Xalan from Promise Repository [12]. At the beginning of the experiment, selected software were partitioned into multiple clusters using PBC. Then the linear regression model has been applied to each cluster to find predicted defects. Finally, to show the importance and effectiveness of the proposed PBC, results are compared with the predic-

tion model considering BorderFlow and the entire system. In this context, by considering the OO classes' relationships and similarities, PBC performs better than the BorderFlow clustering algorithm and the entire system.

3.2 Related Work

The literature in defect prediction focuses on predicting software defects by establishing relationships between software defects and code metrics such as CBO, LCOM, LOC, WMC, NPM, etc. Existing software defect prediction techniques use different types of prediction models, for example, statistical inference and machine learning model, and Datasets such as Promise Repository [12] and NASA Dataset [29] to predict defects. Some prediction models predict defects considering the entire system and other use clustering of source code to predict defects. Here, the widely used defect prediction models using clustering of source code along with the selection of the most significant code metrics are described.

3.2.1 Clustering Based Defect Prediction

The source code clustering based defect prediction model divides the whole software Dataset into multiple clusters for training the prediction model more accurately. The software engineering Datasets always contain lots of variabilities such as heterogeneity among the code metrics. These variabilities cause the poor fit of machine learning algorithms or statistical inferences to the Dataset. If the variablities among the Datasets can be minimized by clustering, it will increase the probability of fitting the data to the machine learning algorithms or statistical inferences. Many researches have already been carried out to group the software engineering Dataset for predicting software defects. Some of those experiments are outlined below.

Schroter et al. proposed a defect prediction model that uses program's import dependencies to predict defect for an OO class [40]. For each OO file, it groups its imported classes to form a cluster and maps the past failure history of those classes to the selected OO class. It then predicts the failure-prone possibility of an OO class by using four prediction techniques based on linear regression model, Ridge regression, Regression tree and SVM respectively. The proposed model has been implemented on 52 eclipse plug-ins. Results show that the design and past failure history of a software can be used in defect prediction. It is the first experiment which tries to group the program's dependencies for predicting defects. Although it works well, the main drawbacks of this experiment is: it only considers import dependencies by ignoring the impact of other dependencies such as call dependencies, data dependencies, etc.

To resolve the above problems, Zimmermann et al. proposed a technique to predict defect at the design time by considering call dependencies, data dependencies, and Windows specific dependencies such as shared registry entries [39]. It uses Support Vector Machine (SVM) to predict the post release defects at design time with precision ranged between 0.58 and 0.73. To perform the experiments, it collects the dependencies of all binaries such as executable files, for example, COM, EXE, etc. and dynamic-link files such as DLL for Windows Server 2003. It concludes that the software's defect proneness can be predicted by using the dependencies among all binaries. These usage of code dependencies indicate the importance of using clustering technique in SDP.

Tan et al. proposed a defect prediction method based on functional clustering of the program to improve the performance [41]. For finding clusters, it uses Latent Semantic Indexing (LSI) to group the software into multiple clusters. It uses the linear regression and logistic regression for building the prediction models. It selects two-thirds of the Dataset to train the prediction models and the remaining for test. The prediction capability is justified by using Pearson and Spearman correlation coefficients of predictive and actual defects [7].

To assess the effectiveness of the proposed model, an experiment has been conducted on a software built by Java with a high fault probability. Results show that the prediction model built on clustering performs better than the class based models in terms of precision and recall. It is another important hypothesis to use the clustering technique in SDP.

The usage of the subtractive clustering algorithm and the fuzzy inference system for early detection of faults was proposed by Sidhu et al. [4]. This approach has been tested with defect Datasets of NASA software projects named as PC1 and CM1. It uses the combined model of requirements metrics and code based metrics from the Dataset. Results show that the accuracy of this model is better than other models considering accuracy, mean square error and root mean square error. Although this approach performs well, there is no defined rules to find the sufficient number of clusters using subtractive clustering algorithm and when to stop executing the algorithm. The prediction model's accuracy shows more researches are needed to perform on source code clustering for defect prediction.

To resolve the variabilities among the Dataset, Menzies et al. proposed a software defect prediction model that learns from software clusters with similar characteristics [5, 6]. It shows learning from software clusters is better than learning from the entire system because it may falsify the data used by the prediction model. It performs clustering by using WHERE clustering technique that considers the code metrics and learning treatment using pairs of neighboring clusters. It also generates rules to reduce the number of defects from the local learning but there also exists the conclusion instability. It advises that empirical software engineering should focus on ways to find the best local lessons for groups of related projects. It also shows that global context is often obsolete for particular local contexts in defect prediction. This premise also shows the importance of using the clustering of software Dataset to provide the accurate Dataset for training SDP model.

Bettenburg et al. [30] proposed three kinds of predictors; those were (i) global model uses the entire Dataset for training; (ii) local model uses the subsets of the Dataset for

training, and (iii) multivariate adaptive regression which splines a global model with local consideration. The third model is the hybrid between global and local model. The proposed three kinds of predictors use linear regression as prediction model. To perform experiments, it collects the Dataset from Promise Repository [12] and then it applies Correlation Analysis (CA) and Variance Inflation (VI) factors analysis on the Dataset to find the potential multi-collinearity between the source code metrics. In the case of local model, Dataset are partitioned into regions by a clustering algorithm, named as MCLUST, based on software code metrics. To avoid over-fitting in the global and local models, the appropriate subset of the independent variables are selected by using Bayesian Information Criterion (BIC). It solves the problems of over-fitting by defining penalty term for each prediction variable entering into the model. Finally it uses 10-fold cross validation to get more stable and robust results. Results show that the local model is better than the global model and the global model with local consideration outperforms both the global and local models in all cases. This is also another implication of using clustering in the defect prediction.

Scaniello et al. [3] proposed a defect prediction model which predicts defects using step wise linear regression (SWLR) that uses clustering of the source code rather than the entire system. It considers references between methods and attributes to form clusters among the related classes using BorderFlow algorithm [42]. The BorderFlow clustering algorithm performs clustering by maximizing the flow from the border to center and minimizing the flow from border to outside of the cluster. Then it applies the SWLR model on each cluster and produces better results than other models that perform prediction considering the entire system. It focuses on clustering using source code whereas Menzies et al. [5, 6] focuses on clustering using code metrics. It forms clusters considering only related classes which means it only uses coupling information among the classes to form clusters. So, the other code metrics' impacts are needed to analyze for defect prediction. It is also another hypothesis to perform more researches on clustering of software Dataset for SDP model.

In a nutshell, a general overview of defect prediction using clustering emphasizes to group the software source code by applying different clustering approaches to train the prediction model more perfectly. All of the above discussed clustering algorithms such as BorderFlow [3], LSI [41], Subtractive clustering algorithm [4] use software code metrics, source code dependencies or code similarities, etc. to group the source codes. Some approaches use PCA to reduce the dimension of the Dataset before applying the different clustering algorithms [4–6]. None of those methods work perfectly in all Promise Repository's Datasets [12]. So, further researches are needed to perform on clustering of source code for defect prediction.

3.2.2 Code Metrics Selection

The code or software metric is a quantitative measure to define the quality of a software system. There are lots of software metrics such as method level, class level, component level, process level, cyclomatic complexity, etc [13]. These software metrics have multidimensional usage in the software industry such as schedule and budget planning, cost estimation, quality testing, software debugging, software performance optimization, etc. Apart from those, it has been used for defect prediction to improve software qualities. Among all the metrics, the method level metric is widely used in structured programming and Object Oriented Programing (OOP) paradigm and the class level code metric is only used for OOP paradigm. All of the above metrics' properties are not significant for defect prediction [13, 43, 44]. To analyze the importance of those metrics, many researches have already been carried out to identify the best set of metrics for defect prediction. Some of these researches are listed below.

Cyclomatic complexity is the first attempt to measure the complexity of a software by Thomas J. McCabe Sr. in 1976 [38]. It computes the complexity of a program by measuring

the linearly independent execution paths from the source code. It draws the control flow graph of a program by considering each instruction as node and data flow among nodes as directed edges. Cyclomatic complexity, (M) of a program can be calculated by using the Equation 3.1.

$$M = EN + 2P \tag{3.1}$$

where,

- E = the number of edges of the graph
- N = the number of nodes of the graph and
- P = the number of connected components

It suggests that the cyclomatic complexity of a program should be smaller than 10 because the high cyclometic complexity means the high probability of having faults [38]. It has been used to measure the quality of source code. Besides this, it is also widely used for test case generation and defect prediction.

Halstead Code Metrics [37] is another way to measure the complexity of a program's source code. It measures the complexity and difficulty of the source code using the number of distinct operators (n_1), the number of distinct operands (n_2), the total number of operators (N_1) and the total number of operands (N_2). It uses n_1, n_2, N_1 and N_2 to calculate the following measures:

- Program vocabulary, $\eta = \eta_1 + \eta_2$
- Program length, N = N_1 +N_2
- Calculated program length, $\hat{\mathrm{N}} = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$
- Volume, V = N $\times$ $\log_2 \eta$

- Difficulty, D = $\frac{\eta_1}{2} \times \frac{N_2}{\eta_2}$
- Effort, E = D $\times$ V
- Programs execution time, T = $\frac{E}{18}$
- Delivered bug, B = $\frac{V}{3000}$

The CK metrics was proposed by Chidamber and Kemerer, to help the designers and managers by providing the inner design details of an OO system [31]. These metrics are calculated by inspecting the relationship among different classes from an OO system. These are Weighted Methods per Class (WMC), Depth Inheritance Tree (DIT), Number of Children (NOC), Coupling Between Object classes (CBO), Response For Class (RFC) and Lack of Cohesion in Methods (LCOM). These metrics are widely employed in fault prediction to improve the quality of an OO system. The detail description of these metrics are given in Chapter 2, Section 2.3.1.

To analyze the impact of the CK metrics in defect prediction [31], Basili et al. performed an experiment using logistic regression to explore the relationship between the CK metrics and the fault-proneness of OO classes [45]. To perform the experiments, it collects eight software projects, developed by eight groups of students at University of Maryland using C++. Results show that the code metrics such as RFC, NOC, DIT are very significant in defect prediction. The CBO, WMC are significant specially in User Interface (UI) level classes and the remaining metric LCOM seems to be significant at all cases.

Gyimothy et al. performed a study to analyze how CK metrics can be employed for the fault-proneness detection of an open source software [46]. For this purpose, it uses statistical methods, for example, logistic and linear regression and machine learning algorithms such as decision tree and neural network. To perform the experiments, it collects an open source software, named as Mozilla and its bug report from Bugzilla database [47]. After that, it applies the statistical methods and the machine learning algorithms to the

Mozilla. Results show that the CBO is the best metric and WMC, RFC and LOC are the most significant metrics, while DIT and LCOM are less significant and the NOC seems to be unimportant in defect prediction.

The usefulness of OO metrics such as CK metric [31], in fault prediction was also investigated by Zhou et al. [44]. It focuses on how accurately the six CK metrics can predict defects when taking the fault severity into account. Each of these metrics have been tested using logistic regression and three machine learning algorithms such as Naive Bayes, Random Forest and Nearest Neighbor with generalization (NNge), on a public NASA Dataset such as KC1. The findings of this experiment show that the CK metrics works well in low severity of defects rather than the high severity. It also shows that the CBO, WMC, RFC and LCOM metrics are statistically significant in both high or low fault severity. The DIT is not significant at any severity level while NOC is only significant at low fault severity.

Pai et al. used Bayesian networks to analyze the effects of CK metrics [31] on the number of defects and the defect proneness of a class [43]. It uses KC1 project from the NASA metrics data repository. It builds a Bayesian network where parent nodes are the CK metrics [31] and child nodes represent the fault content and fault proneness. After the model has been created, it uses Spearman correlation analysis [7] to check whether the variables of the CK metrics are independent or not. It shows that SLOC, CBO, WMC and RFC are the most significant metrics to determine fault content and fault proneness. It discovers that the correlation coefficients of these metrics such as SLOC, CBO, WMC and RFC with fault content are 0.56, 0.52, 0.352, and 0.245 respectively. On the other hand, it also finds that both DIT and NOC are not significant and LCOM seems to be significant for determining fault content.

Catal investigated 90 software defect prediction papers published between 1990 and 2009 [13]. He categorized those papers and reviewed each paper from the perspectives of metrics, learning algorithms and Datasets. According to this review, the method level

metrics such as Halstead [37] and McCabe [38] metrics are most influential metrics in defect prediction and also suggests to use class level metrics for the OO programs. It also shows that the CK metrics is mostly used class level metrics in the defect prediction.

Radjenovic et al. [48] classified 106 papers on SDP according to metrics and context properties. It concludes the proportions of OO metrics, traditional source code metrics, and process metrics are 49%, 27%, and 24%, respectively. Among all of metrics, CK [31] metrics are the most frequently used in defect prediction. The CK metrics has been reported to be more successful than traditional size and complexity metrics.

Okutan et al. performed an experiment to identify the most effective set of metrics in the defect prediction [36]. For this purpose, it uses Bayesian network to determine the probabilistic influential relationships among the software code metrics. It introduces two new code metrics such as Number Of Developers (NOD) and Lack Of Coding Quality (LOCQ). An experiment has been conducted on the 9 open source Promise Repository Datasets [12]. Results show that RFC, LOC and LOCQ are the most significant and effective metrics and LCOM, WMC and CBO are less effective metrics in the defect prediction. In addition, it also shows that NOC and DIT are not effective metrics in defect prediction.

Peng He et al. [35] performed an experiment on 34 releases of 10 open source software projects to find out the most effective set of code metrics for the defect prediction. It also identifies the most suitable defect prediction model that works well in both within project and cross project. The proposed technique uses greedy step wise search algorithm by using forward or backward approach for finding the best of code metrics. It identifies the top k-code metrics that are CBO, LOC, RFC, LCOM, CE (Efferent Coupling), NPM, CBM, WMC, etc. Among all the metrics, it lists CBO, LOC and LCOM as the most significant and influential metrics in defect prediction. It also shows that the success of any prediction model depends on the training data.

The above discussions on the software code metrics show the class level metrics, espe-

cially CK metrics suite, is the most popular and widely used metrics in the software defect prediction. Among the CK metrics, the CBO, RFC, LCOM, WMC are seem to be most significant and NOC and DIT are less significant metrics. The widely used size metrics LOC and NPM are also significant in defect prediction. Many researches also show that the other metrics such as CE and CBM have positive impact in defect prediction. So, to incorporate all the previously investigated experiments, the CK metrics along with LOC and NPM should be used as selected code metrics for the defect prediction.

In summary, in the first section, all existing clustering techniques are summarized, that have been already used in the software defect prediction to improve the training procedure. In the next section, it lists all the most significant code metrics in the context of software defect prediction. This thesis combines both existing work to improve the accuracy and performance of the SDP model.

3.3 Overview of the Java Project

As this research is based on the OO software built by Java, this section provides the detail description of Java project hierarchy to understand the proposed technique. Normally, a Java project consists of some Packages, Classes, Interfaces, etc. For more clarification, the inner details of Class and Package are given below since Interfaces are bit like classes. A Java project hierarchy to represent Project, Packages and Classes is depicted in Figure 3.1.

3.3.1 Class

A class is a building block of all functionality in the OOP. When a class is declared, it actually creates a new data type which is then used to create object of that type. Thus, a class is a template for an object, and an object is an instance of a class.

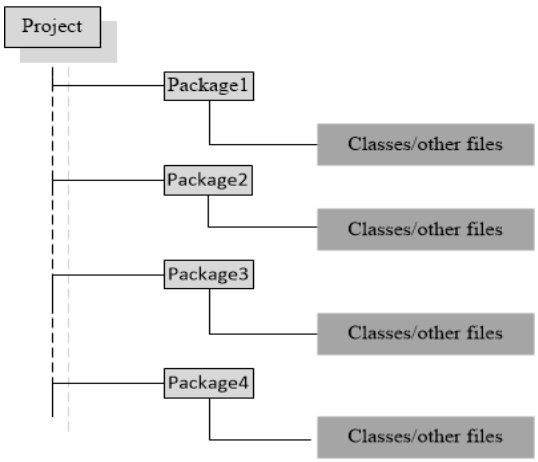

Figure 3.1: The software project hierarchy developed by Java

When a class is declared, it holds its functionalities by specifying the data. A class may contain only data or only some functionalities, but most of real world classes contain both. A class is declared by using the *class* keyword. The general form of a class definition in Java is shown in Source Code 3.1.

Source Code 3.1: The class template in Java

```
class  classname {
        type instance-variable1;
        type instance-variable2;
        // ...
        type instance-variableN;

        type methodname1(parameter-list) {
                // body of method
        }
        type methodname2(parameter-list) {
                // body of method
        }
```

```
        type methodnameN(parameter-list) {
                // body of method
        }
}
```

The data or variables, defined within a class are called instance variables because each instance of the class contains its own copy of those variables. Thus, the data of one object is separated and unique from another. The methods and variables defined within a class are also called members of the class. The instance variables are often used by the method, that decide how the instance variable will be used.

3.3.2 Package

A package is a grouping of related types providing access protection and namespace management. The package command declares a place where the classes will be stored. If package is not declared for a class, the class file will be stored in default package. The general form of the declaring package statement is:

Package *pkg*;

Here, pkg is the name of the package. For example, the following statement creates a package named *myPackage*.

Package *myPackage*;

Java uses file system directory just like a computer's file system directory. More than one file can have the same package name. The source files that are declared must be stored in a package. In the case of nested package, a package may contain inner packages. In that case the package naming structure can be expressed as follows: Package pkg1[.pkg2[.pkg3]]; A short example of Java package is illustrated in Source Code 3.2 and Source Code 3.3.

Source Code 3.2: The package template in Java

```
Package myPackage;
        class  Calculator{
                int  numberA, numberB;

                int  Sum( int  numberA,  int  numberB) {
                        return  numberA+ number;
                }
                int  multiply( int  numberA,  int  numberB) ) {
                        return  numberA* number;
                }
        }
```

Source Code 3.3: The nested package template in Java

```
// Hello .java
import  javax.swing.JApplet;
import  java.awt.Graphics;

public  class  Hello  extends  JApplet {
    public  void  paintComponent(Graphics g) {
        g.drawString(  "Hello , world !", 65, 95);
    }
}
```

3.4 Proposed Package Based Clustering Algorithm

Existing clustering techniques consider a number of source code characteristics, for example, source code dependencies or similarities, code metrics, lexical similarity to group the software into clusters. To the best of author knowledge, no such technique yet considers both code relationship and similarity to group the source codes. In this thesis, a new Package Based Clustering (PBC) algorithm is proposed to group the software using the package information because package holds the related and similar objects in an OO system.

The proposed PBC algorithm groups a software into multiple clusters using related and similar OO classes. The functionality of PBC can be classified as OO Class Identification, Cluster Formation and Cluster Validation. These are summarized below.

3.4.1 OO Class Identification

At the beginning of the clustering process, the proposed algorithm lists down all files from software project and then it identifies potential files that will be considered for constructing clusters. In this context, the software project is built using Java, so the proposed algorithm performs searching by considering the file extension with *.java*.

Software project may contain lots of files such as xml, image, html or other files to make a software functional. Since the OO file determines the behaviors of software, defect prediction model finds only OO files to predict defects. Here, the OO class identification step is performed by searching the files that end with *.java* because software Datasets used in this experiment are developed by Java. The overall OO class identification process for PBC algorithm is illustrated in Algorithm 1.

In this algorithm, it traverse all files using Line 1 and checks the file extension using Line 2. Then it only stores files having extension with *.java* using Line 3. This process stops when no file available for executing works.

Algorithm 1 OO class identification

```
1: for each file do
2:     if fileNameExtention = .java then
3:         Add file to programClassList
4:     end if
5: end for
```

3.4.2 Package Identification and Cluster formation

After the identification of the OO classes, the PBC finds the package name of each class. To identify package information of a Java file, it reads the file and retrieves the package name by matching the pattern structure, for example, package *packageName;*.

At the end of the identification of the package name, it groups the files into multiple clusters based on the package name. For each distinct package name, it creates an array to store the file's name that resides under the same package. If package name is already considered, it adds an OO class to the existing array of that package, otherwise it creates another for the new package to add its OO class. The whole process is performed using Algorithm 2.

Algorithm 2 Package Identification and Cluster formation

```
1: for each programClassList do
2:     Read OO File
3:     Search Each OO File For PackageName
4:     if packageName Contains In PackageContainer then
5:         Add File To packageName
6:     else
7:         Add New PackageName To PackageContainer
8:     end if
9: end for
```

In a word, the PBC finds out the package name of each class and lists all package name from the source codes as shown in Lines $1-2$. If package name is already considered as cluster, it adds OO class to the existing package using Lines $4-5$, otherwise it creates another cluster to add OO class as shown in Lines $6-7$.

3.4.3 Cluster Validation

Packages may contain different number of classes. Some packages may have lots of classes and other may have only few classes. In this thesis, the SDP model uses eight independent or explanatory variables, described in Chapter 2, Section 2.3.1. So if number of classes in a package is less than the explanatory variables used in the prediction model, it would fail to predict defects. In this case, to make defect prediction model successful, small packages are combined to form a joint cluster by applying the Algorithm 3.

Algorithm 3 Cluster validation

```
1: for each Package in packageContainer do
2:     if NumberOfClassInPackage <NumberOfVaribale then
3:         Add To Joint Cluster
4:     end if
5: end for
```

This part only joins small clusters because the prediction model cannot draw conclusion from small clusters. For that purpose, it analyzes each newly created cluster and counts each cluster's elements. Then it combines small clusters to create joint cluster using Line 3 only when the number of elements in a cluster is less than elements than the number of independent variables used in the prediction model.

In a nutshell, the PBC algorithm groups a software project based on package information as package is a collection of similar and related classes. The whole PBC algorithm is illustrated in the Algorithm 4.

3.5 Experimental Setup and Result Analysis

In this section, the experimental environment setup and the results analysis of the PBC along with BorderFlow clustering algorithms are analyzed. This section presents the implementation details of the proposed PBC, Dataset collection, used BorderFlow clustering

Algorithm 4 Package based Clustering (PBC)

Require: Package Container $\mathcal{C} = 0$, Package Name $\mathcal{N} = 0$, Program Class List $\mathcal{L} = 0$, Number Of Varibale $\mathcal{V}$, PackageSize $\mathcal{PS} = 0$, File Name $\mathcal{F}$, Number Of Class In Package, $\mathcal{NP}$.

1: **for each** $file$ **do**
2: **if** $\mathcal{F}$ ends with *.java* **then**
3: Add file to $\mathcal{L}$
4: **end if**
5: **end for**
6: **for each** $\mathcal{L}$ **do**
7: Read program file
8: Search each OO file for $\mathcal{N}$
9: **if** $\mathcal{N}$ contains in $\mathcal{C}$ **then**
10: Add file to $\mathcal{N}$
11: **else**
12: Add new $\mathcal{N}$ To $\mathcal{C}$
13: **end if**
14: **end for**
15: **for each** package in $\mathcal{C}$ **do**
16: **if** $\mathcal{NP} < \mathcal{V}$ **then**
17: Add to joint cluster
18: **end if**
19: **end for**

technique. Finally, the effectiveness of the proposed PBC is measured by taking the mean, variance and standard deviation of absolute residuals.

3.5.1 Software Defect Prediction Model

Software defect prediction model predicts the defects of a class by analyzing the relationships between software code metrics and software defects. In this thesis, the linear regression model is selected as the prediction model to predict defects of a particular class because the relationship among code metrics' properties is linear [7, 49]. The detail description of linear regression model is already described in Chapter 2, Section 2.2.1.

The linear regression model uses a set of independent variables which are WMC, CBO, DIT, LCOM, LOC, NPM, RFC, NOC (see Chapter 2, Section 2.3.1 for detail description) to predict the dependent variable which is defect. The linear equation used by the linear regression model is given in Equation 3.2 [7].

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c \tag{3.2}$$

where,

- Y is the number of defects in an OO class
- $x_1 ... x_8$ are the independent variables which are CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC
- $b_1 ... b_8$ are the coefficient values of those independent variables respectively
- c is the value of Y when all independent variables are 0.

In this thesis, similar as Juban et al. [50], 80% data is used to train the prediction model and the remaining data is used to assess the performance and accuracy of the model.

3.5.2 Prediction Validation

To evaluate the quality of the defect prediction model achieved by the linear regression analysis, the Mean (M), Median (Md) and Standard Deviation (StD) of Absolute Residuals (AR) are computed. The AR value, widely used in the performance measure of the linear regression model [3, 51], is the difference between predicted defects and actual defects of a particular OO class. The smaller value of AR shows the better accuracy of the prediction model [3].

To compare the proposed method PBC with BorderFlow in terms of defect prediction accuracy, the *error* is computed by using the Equation 3.3 [3]. The *error* value shows how much better or worse the proposed dimension reduction technique is in the context of software defect prediction [3].

$$error = \frac{MAR(PBC) - MAR(BorderFlow)}{StDAR(BorderFlow)} \tag{3.3}$$

where,

- MAR(PBC) is the mean value of AR using PBC
- MAR(BorderFlow) is the mean value of AR using BorderFlow
- StDAR(BorderFlow) is the standard deviation of AR using BorderFlow

The *error* value is the ration of two techniques' mean difference and the standard deviation of the technique to which another technique is compared. As a result, it assumes values in between -1 and +1. For a software release, negative values of *error* indicate the proposed approach PBC outperforms BorderFlow technique, while a positive value indicates the BorderFlow outperforms the proposed PBC.

3.5.3 *Existing Clustering Algorithms*

For software defect prediction, many prominent clustering algorithms have been used to group software into multiple clusters. To validate the new clustering algorithm, the comparison needs to be accomplished between the proposed and existing clustering techniques. In this section, the existing clustering algorithm that is considered to compare results with the proposed clustering algorithm is described below.

BorderFlow

BorderFlow clustering algorithm is successfully implemented in [3, 52] to group the software into multiple clusters. It treats the whole software as a collection of nodes to represent the whole software as a graph. It maximizes the flow from the border of each cluster to the nodes within the cluster, while minimizing the flow from cluster to the nodes outside. In this context, the goal of this algorithm is to find groups of tightly coupled classes which are likely to implement a set of related features.

Let, a cluster X, is a subset of V, $b(X)$ is the set of border nodes of X, and $n(X)$ is a function used to identify the set of direct neighbors of X. Ω is a function that assigns the total number of the edges such as dependencies from a subset to another subsets using Equation 3.4. Then BorderFlow ratio can be measured by using Equation 3.5.

$$\Omega(X,Y) = \sum e(c_i,c_j)|c_i \varepsilon X and c_j \varepsilon Y \tag{3.4}$$

$$F(x) = \frac{\Omega(b(X),X)}{\Omega(b(X),n(X))} \tag{3.5}$$

To find group of similar and related classes, this algorithm iteratively selects OO classes known as nodes from $n(X)$ and inserts OO classes in X until $F(X)$ is maximized. The iterative selection of classes ends when $n(X)$ equals to 0 for each set of classes.

3.5.4 Tools and Technologies

In this thesis, the SDP model, the proposed PBC and existing BorderFLow clustering technique have been implemented by using the R [53], C# and Java programming languages respectively. To run R script, Java and C# code, the open source software RStudio, Eclipse and Visual Studio have been used in this experiment [54]. The short description of R, RStudio, Java, Eclipse, C# and Visual Studio are given below.

R: It is a programming language for statistical computing and graphics [53]. It is widely used among statisticians and data miners for data analysis. In this thesis, R has been selected to implement the selected linear regression model because it is open source and easy to implement.

RStudio: To run R script, the open source software RStudio has been used [54]. It is a free and open source integrated development environment for R, written by C++ [54]. RStudio is available in two editions: RStudio Desktop, where the program is run locally.and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server. In this experiment, RStudio Desktop version has been used.

Java: It is a OOP language for building application software. In this thesis, the BorderFlow clustering algorithm is developed using Java.

Eclipse: It is an Integrated Development Environment (IDE) for Java. It contains a base workspace and an extensible plug-in system for customizing the environment. It is written mostly in Java and it can be used to develop applications. In this thesis, the BorderFlow clustering algorithm is performed on the available software using Eclipse.

C#: It is modern, high-level programming language for building apps using Visual Studio and the .NET Framework [55, 56]. It is designed to be simple, powerful, type-safe and OO. In this thesis, the proposed PBC is implemented using C#.

Visual Studio: It is an Integrated Development Environment (IDE) by Microsoft for developing computer programs for Microsoft Windows, as well as web sites, web applica-

tions and web services. In this thesis, the implementation of proposed PBC is performed on the collected Dataset by the help of Visual Studio version 12.

3.5.5 Dataset Collection

The proposed PBC for software defect prediction has been experimented on 8 releases of 2 open source software built by Java. All of those defect Datasets have been downloaded from the Promise Repository [12]. Those Datasets contain the corresponding software code metrics and defect information which are usually used by the prediction model to predict defects for future releases. The source code for Ant and Xalan are downloaded from Apache Repository [57] which are used by PBC and BorderFlow to find clusters from source code. The short description of those Dataset's software are given below:

Ant: It is a library and command line tool for automating software build processes. In this experiment, the Ant releases 1.3 to 1.7 have been selected because its Dataset is widely available and it is developed by Java [12, 57].

Xalan: It is an XSLT processor for transforming XML documents to HTML, text or other XML document types. The available releases from Xalan 2.4 to 2.7 have been considered here [12, 57].

3.5.6 Implementation Details

To perform the experiment, the prediction model using linear regression model, described in Section 3.5.1, is considered as the prediction model here. This prediction model is implemented by using R programming language. This model is also validated using the validation apporach, discussed in Section 3.5.2.

The proposed clustering method PBC is based on the package information for the

Table 3.1: The mean, median and standard deviation of Absolute Residual for PBC

Dataset	BorderFlow			Entire System			PBC		
	MAR	MdAR	StDev	MAR	MdAR	StDev	MAR	MdAR	StDev
Ant-1.7	0.563	0.205	0.789	0.627	0.205	0.933	0.492	0.186	0.861
Ant-1.5	0.117	0.084	0.197	0.260	0.121	0.287	0.199	0.106	0.242
Ant-1.4	0.421	0.325	0.410	0.484	0.493	0.337	0.213	0.213	0.137
Ant-1.3	0.176	0.078	0.340	0.601	0.164	0.794	0.581	0.328	0.799
Xalan-2.7	1.369	1.300	0.391	0.521	0.502	0.404	0.433	0.259	0.498
Xalan-2.6	0.770	0.429	1.062	0.987	0.513	1.103	0.685	0.510	0.645
Xalan-2.5	0.664	0.523	0.568	0.964	0.520	1.543	0.848	0.613	0.873
Xalan-2.4	0.182	0.117	0.358	0.765	0.136	2.137	0.312	0.149	0.502

source code developed using Java, is implemented using C#. The BorderFlow clustering algorithm which finds clusters by maximizing the flow from the border of each cluster to the nodes within the cluster and minimizing the flow from cluster to the nodes outside, is implemented using Java as described in [3].

3.5.7 Result Analysis

This section presents the result analysis of the SDP model using the proposed PBC compared to BorderFlow algorithm and the entire system that considers no clustering technique. The result of the SDP model using different clustering approaches are compared by using the MAR, MdAR and StDAR of the Absolute Residuals (AR). To evaluate the performance of the SDP model using PBC compared to BorderFlow clustering, both PBC and BorderFlow were applied to the Dataset (discussed in Section 3.5.5) to form clusters. Finally, results of proposed PBC are analyzed using the ARs generated from the prediction model.

Table 3.1 highlights the Comparison of MAR, MdAR and StDAR of AR values to identify which method produces smaller AR values. As the smaller AR values represent the better defect prediction model, it is clear that the SDP model considering BorderFlow and PBC algorithms perform better than the entire system in all cases but inconsistencies

Table 3.2: The error values of Dataset

Dataset	Error
Ant-1.7	-8.9%
Ant-1.5	41.6%
Ant-1.4	-50.585
Ant-1.3	118.8%
Xalan-2.7	-238.93%
Xalan-2.6	-7.9%
Xalan-2.5	32.3%
Xalan-2.4	36.3%

exist in AR values produced by the SDP model considering PBC compared to BorderFlow. In some cases, for example Ant-1.7, the SDP model considering PBC performs better than BorderFlow, and for Ant-1.5, the SDP model considering BorderFlow performs better than PBC. From the experimental results, it can be concluded that PBC works well when the package information can accurately group the source codes. In contrary, BorderFlow works well when each cluster get sufficient number of objects so that SDP model gets proper Dataset for training purpose.

The error value shows that how much better or worse the SDP model is compared to other. The error values are calculated from MAR and StDAR values using Equation 3.3 (see Section 3.5.2 for detail description). Usually, the negative error value shows high accuracy and positive value show low accuracy of the SDP model [3]. Since both clustering method perform better than entire system, it is now only needed to show which clustering technique is better in software defect prediction. The error values of PBC is calculated considering only PBC and BorderFlow by applying the Equation 3.3. Table 3.2 summarizes all error values computed by using Equation 3.3 for the PBC clustering technique.

The error values considering the clustering technique PBC and BorderFlow are summarized in Figure 3.2. In this figure, all points under the horizontal line represent the software releases; for these the SDP model has better prediction accuracy. Since, the negative value means that the clustering methods using PBC outperform the clustering methods using

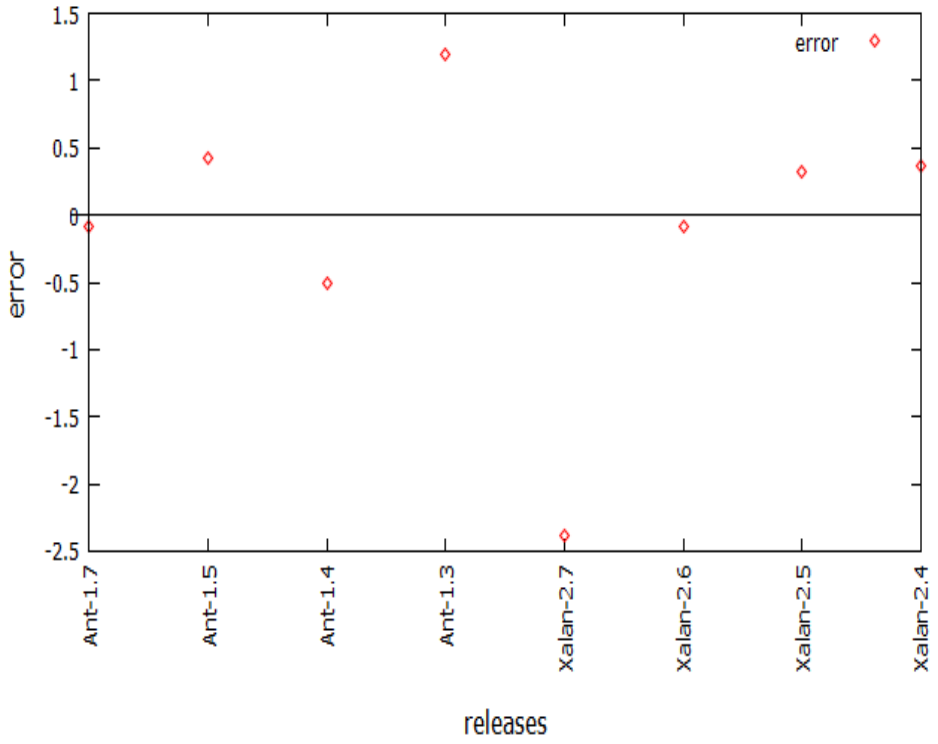

Figure 3.2: The distribution of error values by PBC

BorderFlow. In this figure, 4 Datasets which are Ant-1.5, Ant-1.3, Xalan-2.5 and Xalan-2.4 have positive error values, that means PBC fails to perform better in these Dataset. For Datasets Ant-1.7, Ant-1.4, Xalan-2.6 and Xalan-2.7, the error values are negative which mean PBC performs better in those Datasets because of their proper structure in the development time.

3.6 Conclusion

The proposed clustering technique named as PBC is based on the related and similar OO classes that form packages in java programing convention. It uses textual analysis on source codes to identify OO classes from a software project and lists out those files. To form clusters, it extracts the package information from each OO class by searching the package

name. In special cases, if the number of OO classes of a cluster is smaller than the number of independent variables used in the prediction model, it combines small clusters to enable those for the prediction model. Finally the linear regression model considering PBC is conducted on Ant and Xalan.

Results show that the software defect prediction using the proposed PBC outperforms the prediction models considering the entire system because PBC uses source code similarities and relationships to group the software. For BorderFlow algorithm, the PBC also performs better in some cases. In this context, the prediction model considering PBC performs better in 4 Dataset out of 8 than the prediction models built considering BorderFlow.

The next chapter discusses about the grouping of source code based on the similarity analysis using dimension reduction approach considering the impact of code metrics to number of defects in an OO class to enable the software engineering Dataset for distance based clustering algorithms.

Chapter 4

Similarity Analysis by Reducing the Code Metrics' Dimension

This chapter discusses about the new proposed dimensions reduction technique considering impact analysis of code metrics to defects for the software engineering Dataset. It also explains the used prediction model, selected Dataset, experimental setup and implementation details together with the result analysis of the proposed technique.

4.1 Introduction

Source code similarity analysis by grouping the software into multiple clusters can improve the performance and accuracy of a SDP model. The effectiveness of a SDP model depends on the learning procedure because better learning increases the prediction accuracy of a SDP model. Usually, the software engineering Dataset such as Promise Dataset [12] contains lots of variabilities (for example, heterogeneity among the code metrics), those always hinder the learning procedure of a SDP model. To minimize those variabilities, the heterogeneity among the Dataset needs to be minimum. For that purpose, multiple clustering algorithms can be applied on the Dataset to group heterogeneous data together. Besides those variablities, the Datasets are normally multidimensional because each entry contains lots of code metrics attributes such as CBO, LCOM, RFC, etc. Those make the Dataset multidimensional. Due to the multidimensionality, clustering algorithms may not always perform well. So, if it is possible to minimize the dimensions of the software engineering Dataset based on their similarities, it will definitely help the clustering algorithms.

In any correlation analysis, the dependent variable, for example, number of defects in an OO class, depends on a set of independent variables, for example, CBO, LOC, etc. If

the dimensions of the Dataset are reduced based on the correlation of independent variables to the dependent variable, the new Dataset should get close values for similar objects. The independent variable values can be reduced to two latent variables based on their positive and negative impacts on the dependent variable (for example, defects). In this course, the similarity between objects can be measured based on their Cartesian distance, if the positive and negative latent variables (that is, impact) are plotted in a two dimensional plane.

Software engineering Dataset can be reduced by using various dimension reduction techniques such as Principal Component Analysis (PCA), Factor Analysis (FA), etc. The PCA uses covariance matrix, its eigenvector and eigenvalue for reducing the dimensions. The PCA technique works well for the linear Dataset, but for the nonlinear Dataset and the Dataset having lots of uncorrelated attributes, it cannot reduce the dimensions properly. On the other hand, FA is considered the extension of the PCA. It uses correlation matrix to reduce the Dataset dimensions and the success of FA depends on the choice of the number of factors. Nagappan et al. used PCA to select the best set of attributes for predicting the failure proneness of the software using the code churn and all dependency information [58–60]. As PCA sometimes causes loss of information, the failure of PCA may decrease the performance of the technique. Zimmermenn et al. performed an experiment for predicting defects using network analysis of dependency graphs among various pieces of codes [61]. For that purpose, it uses PCA to select the best set of attributes by reducing the multicollinearity among the Dataset. Since it uses PCA, it also inherits the problems of PCA mentioned above. Menzies et al. used PCA to reduce the multicollinearity of the Dataset for the SDP model [5, 6]. It plots the Dataset considering the greatest variability component in x-axis and the next component in y-axis. It then applies the WHERE clustering algorithm to find the similar objects from the Dataset. Although, it uses only two most significant PCA components for plotting the Dataset, it does not clarify whether only two components can describe all the variances of the Dataset or not.

In this thesis, source code Similarity Analysis by Reducing the Code mertics' dimension (SARCM) is proposed based on the impact of the independent variables to the dependent variable. Where the independent variables are CBO, LCOM, RFC, WMC, etc. (described in section 2.3.1) and dependent variable is the number of defects in an OO class. To identify the relationship between the independent and dependent variable, the regression analysis is applied to the available Dataset to calculate the coefficient values. The coefficient values determine whether the independent variables are positively or negatively significant to the dependent variable. Then the proportionate impact of each independent variable on the dependent variable is calculated by multiplying the coefficient value to the corresponding variable value. Finally, the values that positively significant to the dependent variable are summed and assigned to one variable named as *PosImpactValue* and the values that are negatively significant to the dependent variable are summed to another variable named as *NegImpactValue*. Now the similarity score between two objects (in terms of code metrics) can be measured by the distance where *PosImpactValue* and *NegImpactValue* are considered as x and y respectively. This Dataset is easily plotable to the two dimensional plane considering *PosImpactValue* as x-axis and *NegImpactValue* as y-axis.

As the dimension of the Dataset is reduced based on the significance of independent variables to the dependent variable, the similar objects become closer to each other in the two dimensional plane. Now different distance based clustering technique can accurately identify clusters from the software engineering Dataset with similar properties.

To show the importance and effectiveness of the proposed technique, an experiment has been performed on some open source software such as jEdit, Ant, Xalan, etc. from the Promise Repository [12]. The proposed technique reduces the dimensions of the Dataset for the different distance based clustering algorithms. The clustering algorithms such as DBSCAN [62], WHERE clustering [5, 6] have been applied to the dimension reduced Dataset. Then the linear regression model has been applied to each cluster to find predicted de-

fects. To compare the results of the dimension reduction approach, these two clustering approaches were also applied to another dimension reduced Dataset by PCA. Finally, results are compared to show how dimension reduction technique affects the clustering and the clustering affects the defect prediction model.

Results show that the proposed dimension reduction technique using the coefficient values can successfully assign new values to each entry based on the significance of independent variables to dependent variable. As a result, both DBSCAN [62] and WHERE clustering techniques [5, 6] using SARCM outperform PCA in the defect prediction because PCA may lose some information. Experimental results show that the SDP model outperforms in 14 Datasets out of 17, divided by the DBSCAN and WHERE clustering using SARCM technique.

4.2 Related Work

Due to the multidimensionality of the software engineering Dataset, the distance based clustering algorithms may not work properly. To reduce the multidimensionality of the Dataset, there exists lots of dimension reduction approaches in the literature [63,64]. Many researches in SDP have already used those dimension reduction approaches successfully to reduce the dimensions of the Dataset. In this section, the widely used dimension reduction techniques and their usage in SDP are described.

4.2.1 Existing Dimension Reduction Techniques

Dimension Reduction (DR) is a process of reducing a set of variables into minimum number of variables that can explain the data perfectly. The goal of this approach is to find out a set of correlated variables to form a new smaller set of latent variables with minimum loss

of information [49]. There are lots of ways to perform DR in a Dataset such as Principal Component Analysis (PCA) [63, 65, 66], Feature Selection (FA) [64, 67], etc. Some of the DR techniques are outlined below.

Principal Component Analysis

The Principal Component Analysis (PCA) uses an orthogonal transformation to convert a set of correlated observations into a set of linearly uncorrelated variables called principal components [63, 65, 66]. It reduces the dimensions of a Dataset into minimum number of dimensions that can describe all the varibality of the data. To reduce the dimensions, it first subtract the mean of each dimension from the Dataset and generates covariance matrix of the Dataset. Then it calculates the eigenvector and eigenvalue of the covariance matrix. Finally, it chooses components and feature vector from the eigenvalue and eigenvector. Normally, the total number of principal component is less than or equal to the number of original dimensions of the Dataset. The first principal component is the linear combination of n-variables that has maximum variance, so it gives as much variation in the data as possible. Just like first component, the second principal component is also the linear combination of n-variables maximizing the remaining variation as possible, with the constraint that the correlation between the first and second component is 0. Like first and second component, the i-th principal component maximizes the remaining variation of the data.

For the software engineering Dataset like Promise Repository [12], where each and every attributes are significant to the dependent variable (for example, the impact of CBO, LCOM, LOC, etc.) to the defect, selecting minimum number of components may cause a considerable loss of information. Sometimes PCA analysis selects lots of features to maximize the cumulative variance which also may impede the SDP model because more features make the training process difficult.

Factor Analysis

The Factor Analysis (FA), another way of reducing the dimension of a Dataset [64, 67], is considered as the extension of the PCA technique. It is based on the correlation matrix of the variables whereas PCA uses covariance matrix. It produces latent variables by joining a set of observed variables. To produce latent variables, it uses the available data, common factors and mean of the data. Then it produces the factor model just like regression equation, to produce factor loadings. Finally it transforms the factors into latent variables to minimize the dimensions.

The most critical and important decision for FA is the selection of the number of common factors (m). When the data is normally distributed, the selection of m can easily be measured, but when the number of variables and the sample size is large, the selection of m becomes difficult. For the software engineering Dataset, where attributes are related to each other, the correlation analysis may produce lots of factors and the number of factors depend on the choice of measuring the correlation. The change of correlation measure may change the number of factors. So, multiple interpretation may be possible for FA.

Isomap

Isomap is a dimension reduction method for the nonlinear Dataset [68]. It performs low-dimensional embedding based on the pairwise distance between data points, generally measured by using the Euclidean distance [69]. There are four important steps of the Isomap. These are given below:

- Determine the neighbors of each point.
- Construct a neighborhood graph.
- Compute shortest path between two nodes using Dijkstra's algorithm.

- Compute lower-dimensional embedding.

The connectivity of any point is defined by its nearest k Euclidean neighbors in the neighborhood graph [69]. The success of this method depends on the value of k. If k is too large, it creates the short-circuit errors and if the k is too small, the graph may become too sparse. For software engineering Dataset, where Dataset is not nonlinear all the time. As a result, it is not applicable to the software engineering Dataset having linearity.

4.2.2 Dimensions Reduction in Software Defect Prediction

Dimensions Reduction techniques reduce unimportant and insignificant features from a Dataset. Although, the Dataset having more features contains lots of information, it is difficult to extract information from more features. As a result, the machine learning or statistical models cannot draw conclusions from the data having more features. It also hinders the training process of the machine learning or statistical models. So, it is important to reduce the dimensions of the Dataset by preserving all the variances for the machine learning or statistical models. Many researches in software defect prediction have already used DR techniques. Most prominent of those researches are outlined below.

Nagappan et al. performed an experiment on Windows 2003 to identify the relationship among software dependencies, churn measures and post-release failures [58, 60]. It integrates the call dependencies, data dependencies, architectural dependencies to investigate the propagation of churn across the system. The code churn shows the amount of code change of a software over time. To fit the Dataset into the prediction model, it reduces the multicollinearity among the metrics by using the PCA technique. Then, it uses logistic regression to analyze the software dependence ratio and churn measures as early indicators of failure proneness of a software. Results show that the dependence ratio and churn measure can predict the post release failures and failure-proneness of the binaries. As PCA

analysis selects minimum number of principal components, it may ignore some impact of dependency information and code churn measure. Sometimes, it may select more features to increase the cumulative variance which also may hinder the prediction process of the SDP model, because more features hamper the training process and make the information extraction difficult.

In another experiment, Nagappan et al. proposed a failure prediction model by investigating the relationship between failure-prone software entities and their complexity measures [59]. It uses linear regression analysis as the predictor models for identifying failure prone components. It performed an empirical study on five Microsoft software systems. It shows that multicollinearity exists among the complexity metrics and there is no single set of complexity metrics that could act as a best defect predictor. To overcome the multicollinearity problem, it uses PCA to select minimum numbers of metrics for which the cumulative variance is greater than 96%. After selecting the best set of metrics, it uses these properties to identify the relation among complexity metrics and failure-proneness. Results show that complexity metrics can successfully predict post release defects. Although, the prediction model using PCA works well, but this technique may select lots of features until the cumulative frequencies is greater than 96%. These lots of features may eventually hinder the prediction process of the SDP model because more features hinders the training process.

Zimmermen et al. proposed a defect prediction model using network analysis of dependency graphs among various pieces of code [61]. It uses multiple linear regression analysis as the prediction model for predicting the critical binaries. It uses PCA to reduce the multicollinearity among the Dataset and to select the best set of attributes from the Dataset. It selects only those principal components, for which the cumulative variance is greater than 95%. To show effectiveness of the proposed method, it performed an experiment on Windows Server 2003 and results show that complexity metrics and network measures can

predict 30% and 60% of these critical binaries respectively. As it uses PCA, it may loss small amount of information and may suffer the same problems mentioned above.

Ceylan et al. proposed a defect prediction model using Decision Tree, Multi-Layer Perceptron and radial basis functions to predict the number of total defects per module or function [70]. Before applying these prediction model, it uses PCA to the Dataset to remove multicollinearity from the Dataset by eliminating the correlations among the attributes. The experiment has been carried out on some real life software projects collected from three big software companies in Turkey. Results show that the proposed prediction model improves the performance approximately 32.61% for Company-A and 60% for other two companies. Then it uses mean square error approximation to show the effectiveness of the proposed technique. As it uses PCA, so it also inherits the same problems mentioned above.

Turhan et al. proposed a feature selection model to improve the accuracy of the defect prediction model [71]. It shows the widely used feature selection technique, PCA cannot work in nonlineaer Dataset. To perform feature selection in nonlinear Dataset, it suggested to use Isomap [68]. Hence, the Dataset contains both linear and nonlinear Data, this paper uses both linear and nonlinear feature extraction methods in order to combine information from multiple features. To evaluate the effectiveness of the proposed feature extraction method, several experiments were conducted on the different software projects from NASA repository [29]. This research advices that one should not seek for globally best subset of features, rather to focus on building predictors that combines information from multiple features. It also suggested to use balanced combination of the features before selecting the best set of attributes for the SDP model. Although it uses the combination of PCA and Isomap, it does not mention whether the combination of PCA and Isomap perform well or not, and how much information it losses after applying the combination.

Menzies et al. proposed a software defect prediction model that learns from software clusters with similar characteristics to resolve the variabilities [5, 6]. It performs clustering

of the source code by using WHERE clustering technique that considers only the code metrics and learning treatment using pairs of neighboring clusters. To perform the WHERE clustering in the Dataset, the dimensions of the Dataset are needed to be reduced. It performs PCA to reduce the dimensions of the Dataset. It plots the Dataset considering the most variability component in x-axis and the next component in y-axis. Then, it applies the WHERE clustering algorithm to find out the similar objects. The limitation of this experiment is the consideration of only two PCA's components to plot the Dataset without taking into account others. It does not also mention that whether these two components can describe all the variances or not. As a result, the clustering algorithms considering only two PCA's components may not group the Dataset based on their similarity properly.

For the software defect prediction, sometimes the Dataset needs to be divided into multiple clusters based on their similarity to train the SDP model properly. As the Dataset are multidimensional, the distance based clustering cannot perform well until the dimension is reduced. The existing dimension reduction techniques such as PCA can reduce the dimensions, but when two components are taken into account by avoiding the other components, that causes a great loss of information. So, further research are needed to accomplish to represent the whole Dataset by using only two latent variables so that the distance based clustering algorithms can work well in the context of software defect prediction.

4.3 Proposed Methodology

To segregate the similar OO classes, different clustering algorithms might be applied to the software Dataset to find groups from these. Due to the multidimensionality of the software Dataset, different distance based clustering algorithms cannot work properly. So, to apply different distance based clustering algorithms on the existing Dataset, the dimensions of the Dataset are needed to be reduced. In this dissertation, the dimensions of the Dataset

are reduced based on the significance of independent variables (that is code metrics) to the dependent variable (that is defect). The proposed dimension reduction approach named as SARCM is described below.

4.3.1 Dimension Reduction Approach

In this thesis, the dimension of the software engineering Dataset is reduced by identifying the significance of each of the dimension, for example, CBO, RFC, etc. The significance is measured by the coefficient values of the independent variables to the dependent variable. In this context, the independent variables are CBO, LCOM, RFC, WMC, DIT, LOC, NOC and NPM, and the dependent variable is the number of defects in an OO class. If the dimension reduction technique reduces the dimensions based on the coefficient values of the independent variables, the produced latent variable values will be based on the independent variables' significance to the dependent variables.

The significance of one independent variable to the dependent variable can easily be computed by using regression analysis [7,72]. The regression analysis uses a mathematical equation to show how the value of the dependent variable changes when any one of the independent variables is varied. The relationship of independent variables and the dependent variable can be illustrated by using Equaltion 4.1.

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + ... + b_nx_n + c \tag{4.1}$$

Here, Y is the dependent variable, $b_1...b_n$ are the coefficient values of independent variables, $x_1...x_n$ respectively, those coefficient values represent the amount variable Y changes when variable x_i changes 1 unit and c is the intercept point of y-axis that shows the value of dependent variable when all x's are zero.

Since the used Dataset contains eight independent variables, the Equation 4.1 is mod-

ified to Equation 4.2 for the Dataset. In this context, the regression coefficient shows the significance of CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC to the number of defects in an OO class. To find the regression coefficient, this approach uses the Equation 4.2 as the regression model. Then, this equation is performed on the existing Dataset to find the coefficient value of each independent variable.

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c \tag{4.2}$$

where,

- Y is the number of defects in an OO class
- $x_1...x_8$ are the independent variables such as CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC
- $b_1...b_8$ are the coefficient values of those independent variables respectively
- c is the value of Y when all independent variables are 0.

The coefficient value is a measure of linear association between the variables. The coefficient values can be positive or negative based on the impact of the independent variables to the dependent variable. A coefficient value -

- greater than 0 indicates that two variables are positively related.
- less than 0 indicates that two variables are negatively related.
- 0 indicates that there is no linear relationship between the two variables.

To reduce the dimension of the Dataset, the coefficient values are then multiplied to the corresponding variable values. Then the positive and negative values are summed to produce two latent variables, named as *PosImpactValue* and *NegImpactValue* respectively

for each row in the Dataset. The *PosImpactValue* and *NegImpactValue* are calculated by using the Equation 4.3 and 4.4 respectively. As a result, the final Dataset contains only two values for each entry. The whole procedure to reduce the dimensions of a Dataset is given in Algorithm 5.

$$PosImpactValue = \sum b_i x_i \tag{4.3}$$

where b_i is positive.

$$NegImpactValue = \sum b_i x_i \tag{4.4}$$

where b_i is negative.

Algorithm 5 Dimension Reduction Algorithm

Require: Dataset $\mathcal{D}$, Coefficient Value b, Independent Variable Value $\mathcal{X}$, eps value ε

1: Select a linear regression equation, $Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c$
2: Apply the selected equation on the available $\mathcal{D}$
3: Compute b for each independent variable
4: **for each** $\mathcal{D}$ **do**
5: Set value for $\mathcal{P} = 0$, $\mathcal{N} = 0$
6: **for each** b in each $\mathcal{D}$ **do**
7: Select b for the selected variable
8: **if** $b > 0$ **then**
9: $\mathcal{X} \leftarrow \mathcal{X} \times b$
10: $\mathcal{P} \leftarrow \mathcal{P} + \mathcal{X}$
11: **else**
12: $\mathcal{X} \leftarrow \mathcal{X} \times b$
13: $\mathcal{N} \leftarrow \mathcal{N} + \mathcal{X}$
14: $\mathcal{N} \leftarrow \mathcal{N} \times (-1)$
15: **end if**
16: **end for**
17: Set value $\mathcal{P}, \mathcal{N}$ for the selected entry.
18: **end for**

In the very beginning of this process, a regression equation such as Equation 4.2 is chosen and applied to the existing available Dataset to find the regression coefficients b_i using Lines $1-3$. These regression coefficient values might be positive or negative depending on their impacts to the dependent variable.

Now it iterates to each entry of the Dataset to minimize the dimension by using the Line 4. After selecting each entry, it now iterates for each coefficient value of independent variables by using Line 6. To store the values of the two latent variables, both variables named as $\mathcal{P}$ and $\mathcal{N}$ are initialized by 0.

The significance of each of the independent variables are checked by using Line 6. If the coefficient value has positive impact, this selected coefficient value is multiplied to the corresponding variable value by using Lines $8-9$. The final value is stored to a latent variable $\mathcal{P}$. When the coefficient value of a variable is negative, it means the corresponding variable has negative impact to the dependent variable. Then the negative coefficient value is multiplied to the corresponding variable value by using the Line 11. The new produced value is summed to another latent variable $\mathcal{N}$ by using the Line 12. To avoid the negative sign from the $\mathcal{N}$, this value is multiplied to -1 by using Line 13. Finally, the new dimension value for the selected entry is updated by using Line 17.

4.3.2 Measuring the Similarity of Objects

The proposed dimension reduction technique named as SARCM reduces the dimensions to two, based on the positive and negative significance of independent variables to the dependent variable. As a result, the similar objects get closer values which are *PosImpactValue* and *NegImpactValue* in the dimension reduced Dataset. So, the similarity among the objects can be analyzed using their Cartesian distances in two dimensional plane.

To measure the similarity among the objects, the dimensions of the Dataset is reduced to two latent variables such as *PosImpactValue* and *NegImpactValue* which show the positive and negative impact of independent variables to the dependent variable respectively. When the new Dataset is plotted considering the *PosImpactValue* as x-axis and *NegImpactValue* as y-axis, it distributes the OO classes in such a way that similar classes become closer

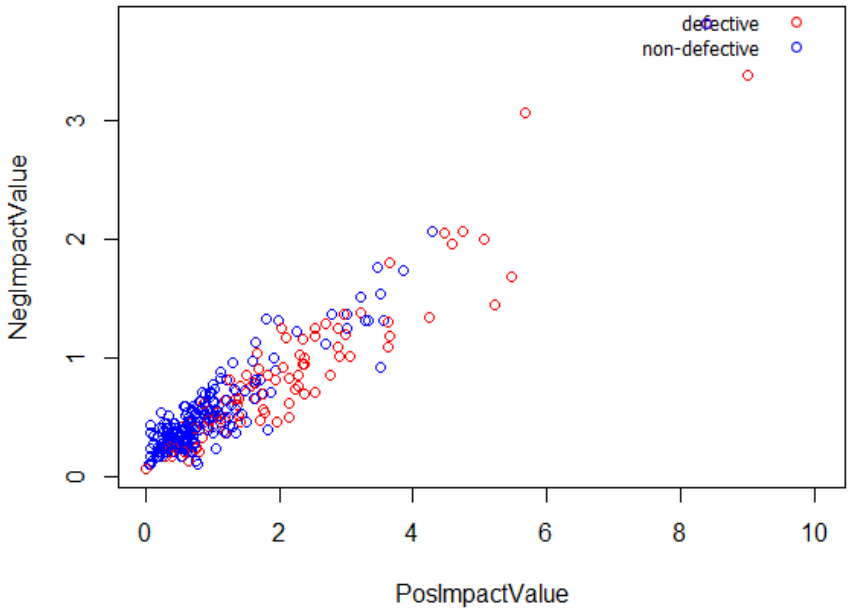

Figure 4.1: The distribution of OO classes in the dimesion reduced Dataset by SARCM for Ant-1.6

to each others and dissimilar classes are farther from each others. In the two dimensional plane, the defective classes are far from y-axis and the non-defective classes are close to y-axis. The dimension reduced Dataset for Ant-1.6 by the proposed technique is illustrated in Figure 4.1.

Figure 4.1 shows the distribution of defective and non-defective classes by using the red and blue circle respectively for the Ant-1.6 project. The goal of this dimension reduction technique is to reduce the dimensions of the Dataset and assign new values based on their significance to the defect. Lets assume a straight line K, from origin (0,0) that divides all objects in the two dimensional plane. In this figure, it is clear that, the density of the red circle is high in the area covered by K and x-axis, because the *PosImpactValue* is high and *NegImpactValue* is low for those objects. On the other hand, the density of blue circle is high in the area covered by K and y-axis, because the *PosImpactValue* is low and

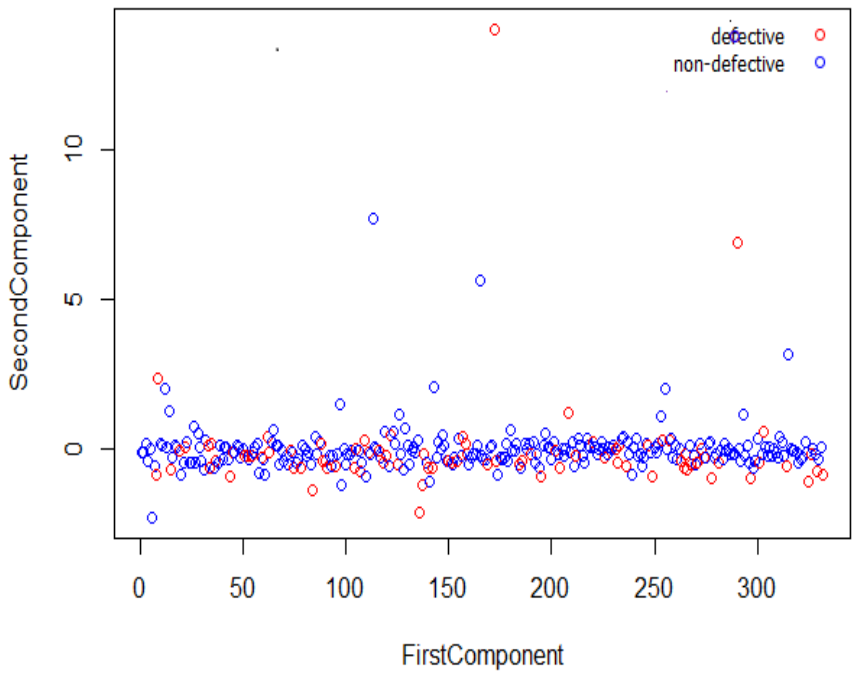

Figure 4.2: The distribution of OO classes in the dimesion reduced Dataset by PCA for Ant-1.6

NegImpactValue is high for those objects. It is now clear that the position of each object in Figure 4.1 corroborates the assumption that the similar objects get closer values in the dimension reduced Dataset and objects are distributed based on their defects.

For the same Ant-1.6, the dimension reduced Dataset by PCA is illustrated in Figure 4.2. In this figure, both defective, represented by the red circle, and non-defective classes, represented by blue circle, are mixed with each other because this technique does not reduce the dimension based on the significance of independent to dependent variable. As a result, similar objects are not closer to each other in this dimension reduced Dataset by PCA.

In terms of distribution of OO class, there exists a significant difference between Figure 4.1 and Figure 4.2. In Figure 4.1, different OO classes create cumulative straight line because SARCM technique reduces the dimension based on the cumulative impact of the

independent variables to dependent variable. In contrary, in Figure 4.2, all OO classes form a line, parallel to x-axis because the value of each entry in the most significant principal component is closer to each others.

As the new dimension reduced Dataset contains closer value for similar objects, the new dimension reduced Dataset can be divided into multiple clusters considering their distance measure. For that purpose, the density based clustering algorithm such as DBSCAN can be a better option. It is also needed to mention that DBSCAN is not the only solution to this approach. Any distance based clustering algorithm can be applied to find clusters from the dimension reduced Dataset. The DBSCAN is considered here, because it is not yet used in SDP. The brief description of the DBSCAN is given below.

DBSCAN

DBSCAN finds clusters from a set of points by measuring their reachability distance. It is a density based clustering approach which groups points that are situated within the density reachability distance, marking as outliers points that lie out of the density reachability distance [62, 73, 74].

In this thesis, as shown above, the dimension reduced Dataset can be plotted in the two dimensional plane where each point represents as an object of an OO system. Lets assume, there exists n objects such as p_1, p_2 ... p_n. Now the DBSCAN can be used to find clusters from these objects by checking their density reachability distance. If the maximum density reachability distance for a Dataset is d, an object p_1 is called density reachable to another object p_n, if and only if, the distance from p_1 to p_2 is equal or less than d, in the same way, the distance from p_2 to p_3 is equal or less than d and so on. Finally, the distance between p_{n-1} and p_n is also less than or equal to d. Then it can be said that P_n is density reachable from p_1. Then it groups objects that are density reachable to one another, marking as

outliers points that lie alone in low-density regions. The clusters, generated by DBSCAN, satisfy the following two properties [62, 73, 74]:

1. All objects within a cluster are mutually density reachable.
2. If an object is density reachable from any object of the cluster, it is part of the cluster as well.

DBSCAN algorithm requires two parameters to classify the Dataset into different clusters. These parameters are:

1. Density reachability distance, generally known as *eps* value, denoted by ε
2. The minimum number of points required to form a dense region, denoted by *minPts*

Density Reachability Distance eps(ε) *and minPts Calculation for DBSCAN*

The DBSCAN clustering algorithm groups the whole software based on the distance measurement. To make the algorithm workable, a density reachability distance, known as *eps* and the minimum object number in a cluster, known as *minPts* are needed to be measured. The *eps* value for DBSCAN, is calculated using the intercept point generated from the prediction model. The *minPts* is calculated from the total number of independent variables because to make the SDP model workable, the total number of observation must be greater than or equal to total number of selected independent variables.

The *eps* value calculation for DBSCAN is most critical and important because the success of the clustering algorithm depends on it. The *eps* value is calculated by summing all intercept values from the prediction model, described in equation 4.5, considering only one independent variable at a time.

$$Y = bx + c \tag{4.5}$$

Since, it uses 8 code metrics, described in 2.3.1, the *eps* value is calculated by summing the eight intercept values considering one independent variable at a time. The whole procedure of calculating the *eps* value for DBSCAN is given in Algorithm 6.

$$eps = \sum b_i \tag{4.6}$$

Algorithm 6 Density Reachability Distance (ε) calculation for DBSCAN

Require: Dataset $\mathcal{D}$, Coefficient Value, b, intercept Point c, eps ε, Independent variable, $\mathcal{X}$
Ensure: ε=0
1: **for each** $\mathcal{X}$ in $\mathcal{D}$ **do**
2: Run $Y = bx + c$ in $\mathcal{D}$
3: Calculate $c = Y - bx$
4: $\varepsilon \leftarrow \varepsilon + c$
5: **end for**
6: **return** ε

In the very beginning of this Algorithm 6, the Equaltion 4.5 is applied to the software Dataset to compute the intercept point c, by using Lines 1-3. This equation is iterated for each of dependent variable assuming the impact of others 0. Then the intercept point for each independent variable is summed to one variable ε by using Line 4.

4.4 Experimental Setup and Result Analysis

In this section, the focus will be kept on how the proposed dimension reduction technique named as SARCM is performed in comparison with the PCA for software defect prediction. This section presents the implementation details of the proposed SARCM along with Selected Prediction Model, Prediction Validation Process, Dataset Collection, Simulation

Environment setup and Implementation Description of the Used Techniques. The proposed SARCM is performed on 6 open source software Datasets available in Promise Repository [12]. Finally, the performance of the SDP model using SARCM is evaluated on the basis of prediction accuracy.

4.4.1 Software Defect Prediction Model and Its Validation

Software defect prediction model predicts the defects of a class by analyzing the relationships between software code metrics and software defects. There exists defect prediction models using various prediction techniques. In this thesis, the linear regression model is selected as the prediction model to predict the defects of a particular class because of the linear the relationship among the code metrics' attributes [7, 49]. The detail description of linear regression model is already described in Chapter 2, Section 2.2.1 and this model has already been used in Chapter 3 for predicting software defects. For more clarification, the prediction model and its validation process is also described here.

The linear regression model uses a set of dependent variables which are WMC, CBO, DIT, LCOM, LOC, NPM, RFC, NOC (see section 2.3.1 for detail description) to predict the dependent variable which is defect. The linear equation used by the linear regression model is given in Equation 4.7 [7].

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c \tag{4.7}$$

where,

- Y is the number of defects in an OO class
- $x_1 ... x_8$ are the independent variables which are CBO, LCOM, DIT, RFC, WMC, LOC, NPM and NOC

- $b_1 ... b_8$ are the coefficient values of those independent variables respectively
- c is the value of Y when all independent variables are 0.

As the success of the linear regression model depends on the training Dataset, after reducing the dimension of the Dataset, the total Dataset is divided into two sets such as trainset and testset. In this thesis, similar as Juban et al. [50], 80% data is used to train the prediction model and the remaining data is used to assess the performance and accuracy of the model.

To evaluate the quality of the defect prediction model achieved by the linear regression analysis, the Mean (M), Median (Md) and Standard Deviation (StD) of Absolute Residuals (AR) are computed. The AR value, widely used in the performance measure of the linear regression model [3, 51], is the difference between predicted defects and actual defects of a particular OO class. The smaller value of AR shows the better accuracy of the prediction model [3].

To compare the proposed dimension reduction approach SARCM with PCA in terms of defect prediction accuracy, the *error* is computed by using the Equation 4.8 [3]. The *error* value shows whether the proposed dimension reduction technique is better or worse in the context of software defect prediction [3].

$$error = \frac{MAR(SARCM) - MAR(PCA)}{StDAR(PCA)} \tag{4.8}$$

where,

- MAR(SARCM) is the mean value of the AR using SARCM
- MAR(PCA) is the mean value of the AR using PCA
- StDAR(PCA) is the standard deviation of AR using PCA

The *error* value, in between -1 and +1, is the ratio of mean difference of two techniques and the standard deviation of the technique to which another technique is compared. For a software release, negative values of *error* indicate the proposed approach SARCM outperforms PCA technique, while a positive value indicates the dimension reduced by PCA outperforms the SARCM.

4.4.2 Tools and Technologies

In this thesis, the SDP model, the proposed technique SARCM, clustering techniques and PCA have been implemented by using the R programming language [53]. To run R script, the open source software RStudio has been used to perform the experiment [54]. The short description of R and RStudio are given below as described in Chapter 3, Section 3.5.4.

R: It is a programming language for statistical computing and graphics [53]. It is widely used among statisticians and data miners for data analysis. In this thesis, R has been selected to implement the selected linear regression model, SARCM and PCA because it is open source and easy to implement.

RStudio: To run R script, the open source software RStudio has been used [54]. It is a free and open source integrated development environment for R, written by C++ [54]. RStudio is available in two editions: RStudio Desktop, where the program is run locally.and RStudio Server, which allows accessing RStudio using a web browser while it is running on a remote Linux server. In this experiment, RStudio Desktop version has been used.

4.4.3 Dataset Collection

The proposed dimension reduction technique for software defect prediction has been experimented on 6 open source software developed using Java. All of these defect Dataset

have been downloaded from the Promise Reposity [12]. These Datasets contain the corresponding software code metrics and defect information which are usually used by the prediction model to predict the defect for future releases. The short description of these Dataset's software are given below.

Ant: It is a library and command line tool for automating software build processes. In this experiment, the Ant releases 1.3 to 1.7 have been selected because its Dataset is widely available and it is built using Java [12].

jEdit: It is a text editor for programmers, built using Java. It supports more than 200 file types. The available jEdit version 4.2 and 4.3 have been considered here [12].

Xalan: It is an XSLT processor for transforming XML documents to HTML, text or other XML document types. The available releases from Xalan 2.4 to 2.7 have been considered here [12].

Camel: It is a rule-based routing and mediation engine, built using Java, which provides enterprise integration patterns using an API to configure routing and mediation rules. The version 1.0 to 1.7 have been considered in this course [12].

Synapse: It is an enterprise service software that provides support for XML and SOAP. In this thesis, the only available Synapse 1.2 Dataset has been considered [12].

Tomcat: It is an open-source web server and servlet container developed by the Apache Software Foundation. In this thesis, the available tomcat Dataset has been downloaded from Promise Repository [12].

4.4.4 Implemented Clustering Approaches

The SARCM technique reduces the dimensions to two latent variables which are *PosImpactValue* and *NegImpactValue* based on the positive and negative significance of independent variables to the dependent variable respectively. Now the dimension reduced Dataset is

plotted into two dimensional plane considering *PosImpactValue* as x-axis and *NegImpact-Value* as y-axis which makes closer the similar objects. Then the two clustering approaches which are DBSCAN and WHERE clustering approaches can be used to find the most similar objects. The detail description of DBSCAN is given in Section 4.3.2 and the description of WHERE clustering approach is given below.

WHERE clustering: It finds the software artifacts with similar properties by using FASTMAP heuristic [5,6]. Given n objects plotted in the two dimensional plane, the greatest variability of two furthest objects are found as follows.

1. Pick any object Z at random;
2. Find the object X that is furthest away from Z;
3. Find the object Y that is furthest away from X.

If each object now has a distance a, to the origin $(0,0)$ and distance b, to the most remote object. From the Pythagoras and cosine rule, each object is at the point (x,y) can be measured by using Equation 4.9 and Equation 4.10 respectively.

$$x = \frac{(a^2 + c^2 - b^2)}{2c} \tag{4.9}$$

$$y = \sqrt{a^2 - x^2} \tag{4.10}$$

Now it calculates the median values of each dimension $(\hat{x},\hat{y})$ and divides the whole plane into: *NorthWest*, *NorthEast*, *SouthWest*, *SouthEast* regions considering the $(\hat{x},\hat{y})$ as the center point.

4.4.5 Implementation Description of the Used Techniques

To perform clustering algorithms on the Dataset, a tool based on the aforementioned dimension reduction technique named as SARCM was implemented to make Dataset two-

dimensional. The proposed dimension reduction technique based on the impact of independent variable to dependent variable was implemented using R script [53].

After reducing the dimensions of the above Dataset by SARCM, the prominent clustering algorithms such as DBSCAN [62, 73] and WHERE clustering [5, 6] approaches were applied on the dimension reduced Dataset. The DBSCAN clustering [62] which finds clusters from a Dataset based on the reachability among multiple objects, was implemented using R language [53]. The detail description of DBSCAN is given in Section 4.3.2.

The WHERE clustering algorithm which finds the software artifacts with similar properties by using FASTMAP heuristic, was implemented using R [5, 6]. This clustering technique uses Pythagorus and cosine rule to select a point to divide the all objects, plotted in two dimensional plane, into *NorthWest*, *NorthEast*, *SouthWest*, *SouthEast* regions.

The SDP model using linear regression analysis to predict defects of an OO class using the relationship between code metrics and software defects, was implemented using the R script [53]. Then, this prediction model was applied to clusters of the Dataset found by the above clustering techniques to predict defects for an unknown OO class.

4.4.6 Result Analysis

This section presents the implementation schemes of the proposed dimension reduction technique SARCM along with its impact on software defect prediction. The proposed SARCM was implemented on 6 open source software, available in Promise Repository [12]. The performance of SARCM is compared with PCA to evaluate which technique works well in the context of software defect prediction. Finally, the performance of SARCM is evaluated on the basis of prediction accuracy using absolute residual values.

The result of the selected SDP model using different clustering approaches are compared by using the MAR, MdAR and StDAR of the Absolute Residuals (AR). To evaluate

the performance of SARCM compared to PCA, two clustering techniques which are DBSCAN and WHERE are applied separately to the dimension reduced Dataset. For the Dataset, dimension reduced by SARCM, both DBSCAN and WHERE techniques divide the whole Dataset into multiple clusters based on the similarity of the OO classes, because the SARCM technique reduces the dimension in a way where the similar objects get closer values. For the Dataset, dimension reduced by PCA, the above mentioned clustering methods also divide the Dataset into multiple clusters based on the variances among elements, because PCA reduces the dimensions in such a way that the selected principal component can describe all variances. Then the SDP model uses those clusters of Dataset for training purpose to predict software defects. Finally the performance of the SDP model is measured by taking the residual value of predicted and actual defects.

The analysis is divided into two phases. In phase I, the comparison of AR Values are analyzed by computing the MAR, MdAR and StDAR values. In phase II, the error values are calculated to determine which technique performs well in software defect prediction.

Phase I: The Comparison of Absolute Residual Values

The Absolute Residual is the difference between actual and predicted defects. The low value of AR shows the high performance and accuracy of a SDP model, and in contrary, the high value determines the low performance and accuracy of a SDP model [3]. In this thesis, the descriptive statistics of the ARs are summarized in Table 4.1 to compare the impact of the proposed SARCM technique on the SDP model. This table illustrates the comparison of MAR, MdAR, StDAR values of DBSCAN and WHERE clustering techniques using the dimension reduced Dataset by SARCM and the PCA respectively. The descriptions of DBSCAN using SARCM vs. DBSCAN using PCA and WHERE using SARCM vs. WHERE using PCA are given below.

Table 4.1: The mean, median and standard deviation value of AR

Dataset	DBSCAN using SARCM			DBSCAN using PCA			WHERE using SARCM			WHERE using PCA		
	MAR	MdAR	StDev	MAR	MdAR	StDev	MAR	MdAR	StDev	MAR	MdAR	StDev
Ant-1.7	0.46	0.24	0.54	0.37	0.18	0.54	0.65	0.48	0.60	0.71	0.30	1.07
Ant-1.6	0.48	0.29	0.59	0.76	0.27	1.48	0.99	1.02	0.70	0.99	0.63	0.98
Ant-1.5	0.13	0.10	0.12	0.13	0.08	0.14	0.30	0.05	0.67	0.40	0.14	0.47
Ant-1.4	0.27	0.19	0.22	0.32	0.29	0.18	2.85	0.17	7.68	0.64	0.45	0.57
Ant-1.3	0.25	0.15	0.31	0.31	0.08	0.49	0.39	0.17	0.50	1.06	0.26	2.82
Xalan-2.7	0.664	0.57	0.40	1.11	0.37	3.60	0.38	0.26	0.37	0.62	0.62	0.45
Xalan-2.6	0.72	0.57	0.54	0.87	0.62	0.88	1.14	0.94	0.93	1.52	0.89	1.85
Xalan-2.5	0.62	0.53	0.51	0.78	0.61	0.87	0.83	0.56	0.88	0.80	0.51	0.81
Xalan-2.4	0.28	0.12	0.45	0.25	0.13	0.33	0.30	0.16	0.44	0.33	0.15	0.47
Synapse-1.2	0.67	0.37	0.69	0.55	0.46	0.45	0.61	0.38	0.62	1.0	0.86	0.86
jEdit-4.3	0.02	0.02	0.02	0.04	0.01	0.12	0.10	0.04	0.15	0.16	0.08	0.19
jEdit-4.2	0.21	0.158	0.202	0.349	0.313	0.415	0.397	0.288	0.350	0.28	0.13	0.41
jEdit-3.2	3.01	1.97	4.340	6.53	2.23	10.02	2.15	1.34	3.23	1.94	1.33	2.04
Camel-1.6	0.748	0.322	1.035	1.549	0.490	7.064	1.065	0.403	1.340	1.23	0.64	1.50
Camel-1.4	0.69	0.27	1.34	0.86	0.51	0.90	0.71	0.23	1.33	1.45	0.42	2.58
Camel-1.2	1.11	0.53	1.27	1.20	0.62	2.21	1.34	0.70	1.41	1.50	1.00	1.73
Tomcat	0.149	0.046	0.241	0.290	0.093	0.406	0.189	0.033	0.303	0.189	0.126	0.274

DBSCAN using SARCM vs. DBSCAN using PCA

In Table 4.1, the column named DBSCAN using SARCM and DBSCAN using PCA show that MAR values are minimum for DBSCAN using SARCM. It means that the prediction model considering DBSCAN using SARCM produces smaller AR values compared to DBSCAN using PCA. Which indicate that the SDP model considering DBSCAN using SARCM can accurately predict defects for the OO classes. In this table, MAR values are high for Xalan-2.7 and Camel-1.6. For the Dataset, Xalan-2.7, the MAR values of DBSCAN using SARCM and DBSCAN using PCA are 0.664 and 1.11 respectively and for Dataset, Camel-1.6, the values are 0.748 and 1.549 respectively. So, the accuracy of the prediction model is low for those Dataset and the MAR value is minimum for DBSCAN using SARCM which indicates that DBSCAN using SARCM performs better than DBSCAN using PCA. For the Dataset, jEdit-4.3, the MAR values of DBSCAN using SARCM and DBSCAN using PCA are low and these are 0.02 and 0.04 respectively. Here, the MAR value of DBSCAN using SARCM is also smaller than DBSCAN using PCA which means that the accuracy of the prediction model considering DBSCAN using SARCM is higher than using PCA. For the Dataset Ant-1.5, both techniques produce equal MAR value of 0.13, which indicates that the accuracy of the prediction model is equal in both cases.

This technique fails in only 3 Datasets which are Ant-1.7, Synapse-1.2 and Xalan-2.4, because for those Datasets, DBSCAN cannot divide those Datasets properly. The detail inspection in those Dataset give that those contain low number of objects, so the clustering process produces clusters with small number of objects. As a result, SARCM cannot perform well in those Datasets that result low prediction accuracy in SDP model. In a nutshell, the SDP model considering SARCM based DBSCAN outperforms in 14 Datasets out of 17, listed in Table 4.1, which indicates that SARCM based DBSCAN is better than PCA based DBSCAN in the context of software defect prediction.

WHERE using SARCM vs. WHERE using PCA

For WHERE clustering method, illustrated in column named WHERE using SARCM and WHEHE using PCA (Table 4.1), the mean value of AR is also minimum in WHERE using SARCM compared to DBSCAN using PCA in 14 Datasets out of 17 which is an indication of better prediction accuracy of SDP model considering WHERE using SARCM. For Ant-1.3, the MAR values of WHERE using SARCM and WHEHE using PCA are 0.39 and 1.06 respectively. For Camel-1.4, the MAR values of SARCM based WHERE and PCA based WHEHE are 0.71 and 1.45 respectively. Since, the AR value is high in Ant-1.3 and Camel-1.4, the accuracy of the prediction model using SARCM is low in those two Datasets. For Dataset Ant-1.6, both techniques produces equal MAR value of 0.99, which indicates that both DBSCAN using SARCM and DBSCAN using PCA work well in this Dataset. The minimum values 0.10 and 0.16, are produced in SARCM based WHERE and PCA based WHERE respectively for jEdit-4.3. Here, the MAR value of SARCM based WHERE is less than PCA based WHERE, which indicates that the prediction model considering SARCM based WHERE performs better than PCA based WHERE.

In a nutshell, the SDP model considering WHERE using SARCM outperforms in 14 Datasets out of 17 software Dataset, listed in Table 4.1. This technique fails in 3 Datasets, which are Ant-1.4, jEdit-4.2 and Xalan-2.4, among these, the MAR value is high for Ant-1.4 which means that it largely fails in this Dataset. The detail inspection in those Dataset gives that those contain low number of objects, so the clustering algorithms produce clusters with small number of objects. As a result, SARCM cannot perform well in those Datasets that result low prediction accuracy of the SDP model.

Phase 2: The Comparison of Error Values

The error value shows how much better or worse the SDP model is compared to other. The error values are calculated from MAR and StDAR vales using Equation 4.8 (see Section 4.4.1 for detail description). Usually, the negative error value shows high accuracy and

positive value shows low accuracy of the SDP model [3]. The error values of DBSCAN are calculated considering DBSCAN using SARCM and DBSCAN using PCA by applying the Equation 4.8. In the same way and using the same equation, the error values for WHERE clustering are also calculated considering WHERE using SARCM and WHERE using PCA. Table 4.2 summarizes all error values computed by using Equation 4.8 for the DBSCAN and WHERE clustering techniques.

Table 4.2: The error values of all Datasets for DBSCAN and WHERE

Dataset	ERROR	
	DBSCAN	WHERE
Ant-1.7	16%	-6%
Ant-1.6	-18.87%	.135%
Ant-1.5	-.576%	-20.58%
Ant-1.4	-31.31%	383%
Ant-1.3	-12.38%	-23.4%
Xalan-2.7	-12.52%	-52.23%
Xalan-2.6	-17.40%	-20.29%
Xalan-2.5	-0.18.65%	4.7%
Xalan-2.4	8.6%	-6.7%
Synapse-2.2	26.5%	-44.63%
jEdit-4.3	-13.23%	-29.27%
jEdit-4.2	-32.85%	26.82%
Camel-1.6	-11.3%	-11.55%
Camel-1.4	-18.72%	-28.546%
Camel-1.2	-4.42%	-8.82%
Camel-1.0	-17.76%	-14.8%
Tomcat	-.34.7%	-0.25%

The error values considering the clustering technique DBSCAN are summarized in Figure 4.3. In this figure, all points under the horizontal line represent the software releases, for those the SDP model has better prediction accuracy. Since, the negative value means that the clustering methods using SARCM outperform the clustering methods using PCA, the clustering method DBSCAN using SARCM outperforms the clustering method DBSCAN using PCA in 14 Datasets. In this figure, only 3 Datasets which are Ant-1.7, Xalan-2.4

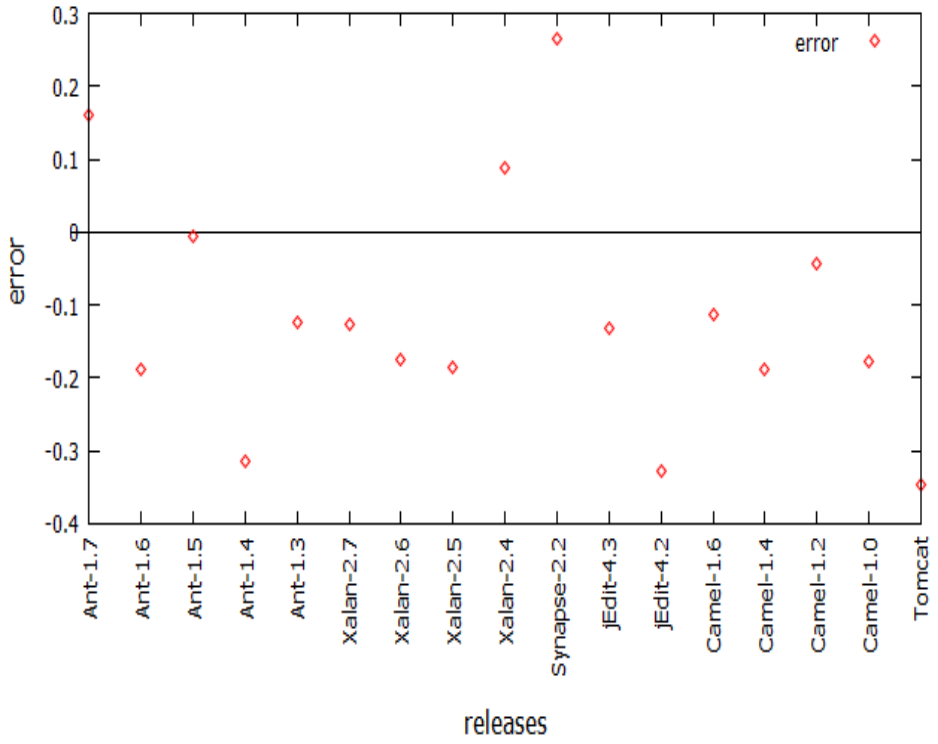

Figure 4.3: The distribution of error values for DBSCAN

and Synapse-1.2 have positive error values, so the clustering algorithm DBSCAN using SARCM fails in those Datasets because of the inappropriate reachablity distance measure for DBSCAN. As a result, the DBSCAN cannot properly group the software Dataset which actually results low prediction accuracy of the SDP model.

In the same way, the error values considering the WHERE clustering technique are summarized in Figure 4.4. In this figure, all points under the horizontal line represent the software releases for those the SARCM based WHERE clustering method outperforms the PCA based WHERE clustering method. The points above the horizontal line represent the software releases for those PCA based WHERE clustering methods outperforms SARCM based WHERE. Here only 3 Datasets reside above the horizontal line and the remaining 14 Datasets reside under the line. So, the WHERE clustering algorithm considering SARCM outperforms PCA in 14 Datasets. It largely fails in Ant-1.4, because it contains lower num-

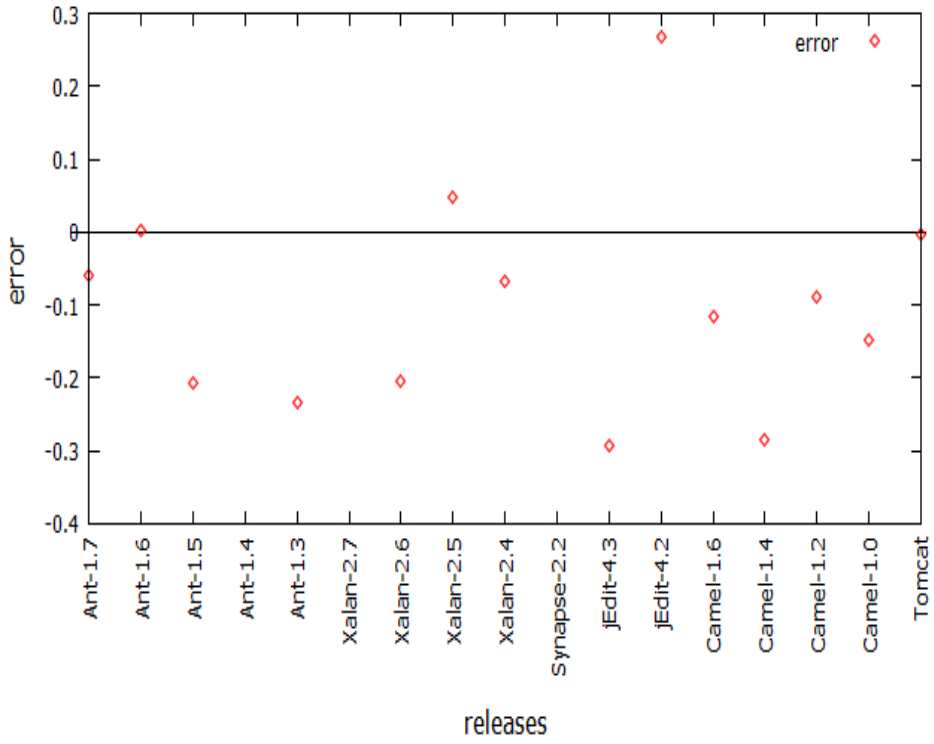

Figure 4.4: The distribution of error values for WHERE

ber of objects and after applying clustering technique, it produces multiple clusters with small amount of objects which actually results low prediction accuracy of the SDP model. For Xalan-2.5 and jEdit-4.2, the error values are close to zero, so it can be negligible.

The SDP model has better prediction accuracy when the clustering algorithms use the dimension reduced Dataset by SARCM, because SARCM reduces the dimension based on the significance of CBO, RFC, LCOM, WMC, DIT, NPM, LOC and NOC to the software defects. As a result, the dimension reduced Dataset gets closer value for similar objects and SDP model gets similar Dataset for learning procedure. In principle, it can be said that the proposed SARCM can significantly improve the accuracy of SDP model which results lower AR values compared to PCA.

4.5 Conclusion

In this chapter, a new approach named as SARCM, to reduce the dimensions for the software engineering Dataset has been proposed. SARCM reduces the dimensions of Dataset using the significance of the independent variables which are CBO, RFC, LCOM, WMC, DIT, NPM, LOC and NOC to the dependent variable which is the number of software defects. The dimension reduced Dataset by SARCM usually gets closer two dimensional values for similar objects. If the dimension reduced Dataset is plotted in the two dimensional plane, the similar objects situate closer to each other and the dissimilar objects are far from each others. As a result, multiple distance based clustering techniques can accurately group the software into multiple clusters based on their similarities. The success of grouping software into multiple clusters depends on clustering techniques, SARCM provides the platform where multiple clustering algorithms can work.

In this thesis, to evaluate SARCM in the software defect prediciton, two clustering algorithms which are DBSCAN and WHERE have been used. These two methods divide the whole Dataset into multiple clusters so that the SDP models get similar Dataset for learning purpose. Results show that the SDP model performs well in the 14 Datasets out of 17, considering SARCM based clustering compared to PCA based clustering .

The next chapter discusses about the overall summary of the proposed PBC and SARCM along with the further scope for improvements of proposed solutions.

Chapter 5

Discussion and Conclusion

In this chapter, discussion regarding the proposed PBC and SARCM will be presented followed by the future direction and scope for improvements of the proposed solutions.

5.1 Introduction

The success of software defect prediction model largely depends on how better the model learned from the software Dataset such as code metrics and past defect information. Since the Dataset having lots of variabilities impede the accuracy and performance of the prediction model, minimizing those variabilities may improve the performance. Besides this, multidimensionalities among the Dataset thwart the clustering methods to perform properly on these Dataset. In this thesis, two different approaches for grouping the source code based on their similarities are proposed. Firstly, Package Based Clusering (PBC) that uses package information to find the similarity among the software. Secondly, Similarity Analysis by Reducing the Code Metrics (SARCM) that aims to reduce software engineering Datasets so that any distance based clustering can work on these Datasets.

5.2 Package Based Clustering

Package Based Clustering (PBC) uses package information to group the source code by Java programming convention. Here, PBC is used to combine the related objects from the software built by Java such that the whole software is divided into multiple clusters based on package information. As a result, the defect prediction model gets useful chunk of Dataset for training purposes.

The proposed technique has been experimented on 8 releases of 2 open source software which are Ant and Xalan [12, 57]. Besides PBC, the well known BorderFlow algorithm was also applied to those Dataset for performance comparison. Finally, the SDP model considering PBC, BorderFlow and the entire system were implemented and compared. Results show that the PBC based SDP model performs better than other approaches.

5.3 Similarity Analysis by Reducing the Code Metrics' Dimension

Similarity Analysis by Reducing the Code Metrics' dimension (SARCM) is proposed for reducing dimensions for software engineering Dataset through impact of code metrics such as CBO, LCOM, etc. to the number of software defects in an OO class. The impacts of CBO, RFC, LCOM, WMC, etc. to defect are measured by using regression analysis which is a component of SARCM. The regression coefficient acts as an impact of code metrics to the number of software defects.

Now the dimension of Dataset is reduced to two variables which are *PosImpactValue* and *NegImpactValue*, based on the positive and negative impact of the code metrics to the number of software defects respectively. This dimension reduced Dataset is plotted on the two dimensional plane considering *PosImpactValue* as x-axis and *NegImpactValue* as y-axis for the distance based clustering apporaches. In this thesis, the DBSCAN and WHERE clustering techniques have been applied to dimension reduced Dataset to group those into multiple clusters based on their closeness.

Finally, the SDP model has been applied to 17 releases of 6 open source software developed using Java. During the experimentation, the SDP model learned from the clusters of the Dataset by DBSCAN and WHERE clustering techniques. To compare the results, both clusters were also applied to the Dataset dimension reduced by PCA. Then the same

SDP model was inherited and implemented on Dataset dimensioned reduced by PCA using DBSCAN and WHERE. Finally, results show that the SDP model performs well in the 14 Datasets out of total.

5.4 Future Research Directions

This research aims at improving the accuracy and performance of SDP models through improving the learning procedure. The improvement is made through introducing similarity analysis to the Dataset. However, this research can be extended to following directions.

Code Metrics Selection

There are more software code metrics in the literature to define quality of a software. In this experiment, 8 most prominent code metrics have been used for reducing the dimensions of the software engineering Dataset to two latent variables. Further researches can be done to incorporate other code metrics' impact to the dimension reduction technique.

Additional Clustering techniques

In the field of software defect prediction, the proposed PBC and well known clustering techniques WHERE and DBSCAN are used to group the source code into multiple clusters. PBC works at the source code level to find clusters and other two techniques work at code metrics level, generated from source code, to group software into multiple clusters. Other clustering techniques (for example, K-means, OPTICS, etc.) can be experimented on those dimension reduced Dataset to address their significance on the software defect prediction.

Dataset Collection

Here, experiments were performed on the 17 releases of 6 open source software collected from Promise Repository [12]. This repository contains other software Dataset and researchers are continuously adding Dataset into it. Further experimentation can be accomplished on the new Dataset available in the same repository. Although this experiment has

been conducted on the open source software, there is also a scope to incorporate this thesis into real life software projects by working collaboratively with software companies.

5.5 Final Remarks

This thesis has demonstrated that training the SDP model by using similar Dataset can improve the performance and accuracy. The similarity among objects can be analyzed based on their variablities using clustering techniques. It has found that reducing multidimentionalities among the Datasets helps the clustering algorithms to perform more accurately.

Bibliography

[1] G. Tassey. The economic impacts of inadequate infrastructure for software testing, 2002.

[2] Roger S. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education, 5th edition, 2001.

[3] Giuseppe Scanniello, Carmine Gravino, Andrian Marcus, and Tim Menzies. Class level fault prediction using software clustering. In *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE),2013*, pages 640–645. IEEE, 2013.

[4] Ramandeep S Sidhu, Sunil Khullar, Parvinder S Sandhu, RPS Bedi, and Kiranbir Kaur. A subtractive clustering based approach for early prediction of fault proneness in software modules. *World Academy of Science, Engineering and Technology*, 67, 2010.

[5] Tim Menzies, Andrew Butcher, David Cok, Andrian Marcus, Lucas Layman, Forrest Shull, Burak Turhan, and Thomas Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, 39(6):822–834, 2013.

[6] Tim Menzies, Andrew Butcher, Andrian Marcus, Thomas Zimmermann, and David Cok. Local vs. global models for effort estimation and defect prediction. In *26th IEEE/ACM International Conference on Automated Software Engineering Proceedings of the 2011*, pages 343–351. IEEE Computer Society, 2011.

[7] Jacob Cohen, Patricia Cohen, Stephen G West, and Leona S Aiken. *Applied multiple regression/correlation analysis for the behavioral sciences*. Routledge, 2013.

[8] Freda Kemp. Applied multiple regression/correlation analysis for the behavioral sciences. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 52(4):691–691, 2003.

[9] David A Freedman. *Statistical models: theory and practice*. Cambridge University Press, 2009.

[10] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[11] Finn V. Jensen. *Introduction to Bayesian Networks*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1996.

[12] Tim Menzies, Bora Caglayan, Zhimin He, Ekrem Kocaguneli, Joe Krall, Fayola Peters, and Burak Turhan. The promise repository of empirical software engineering data, June 2012.

[13] Cagatay Catal and Banu Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[14] K Soundarraj and T Parthasarathi. Major reasons for bugs in software applications.

[15] James S. Huggins. First computer bug. `http://www.jamesshuggins.com/h/tek1/first_computer_bug.htm` , 2015, (accessed on April 29, 2015).

[16] Frederick P Brooks Jr. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India, 1995.

[17] Mary Jean Harrold. Testing: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 61–72. ACM, 2000.

[18] Luay Ho Tahat, Boris Vaysburg, Bogdan Korel, and Atef J Bader. Requirement-based automated black-box test generation. In *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, pages 489–495. IEEE, 2001.

[19] David R Cox. The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 215–242, 1958.

[20] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

[21] John C. Platt. Advances in kernel methods. chapter Fast Training of Support Vector Machines Using Sequential Minimal Optimization, pages 185–208. MIT Press, Cambridge, MA, USA, 1999.

[22] Terrence S Furey, Nello Cristianini, Nigel Duffy, David W Bednarski, Michel Schummer, and David Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[23] James M Keller, Michael R Gray, and James A Givens. A fuzzy k-nearest neighbor algorithm. *IEEE Transactions on Systems, Man and Cybernetics*, (4):580–585, 1985.

[24] Thierry Denoeux. A k-nearest neighbor classification rule based on dempster-shafer theory. *IEEE Transactions on Systems, Man and Cybernetics*, 25(5):804–813, 1995.

[25] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *Database TheoryICDT99*, pages 217–235. Springer, 1999.

[26] Ronald R Hocking. A biometrics invited paper. the analysis and selection of variables in linear regression. *Biometrics*, pages 1–49, 1976.

[27] Leonard E Baum and Ted Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, pages 1554–1563, 1966.

[28] Iker Gondra. Applying machine learning to software fault-proneness prediction. *J. Syst. Softw.*, 81(2):186–195, February 2008.

[29] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Nasa metric data program, June 2007.

[30] Nicolas Bettenburg, Meiyappan Nagappan, and Ahmed E Hassan. Think locally, act globally: Improving defect and effort prediction models. In *9th IEEE Working Conference on Mining Software Repositories (MSR), 2012*, pages 60–69. IEEE, 2012.

[31] Shyam R Chidamber and Chris F Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994.

[32] Jagdish Bansiya and Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on Software Engineering*, 28(1):4–17, 2002.

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

[34] Fernando Brito Abreu and Rogério Carapuça. Object-oriented software engineering: Measuring and controlling the development process. In *Proceedings of the 4th international conference on software quality*, volume 186, 1994.

[35] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology*, 59:170–190, 2015.

[36] Ahmet Okutan and Olcay Taner Yıldız. Software defect prediction using bayesian networks. *Empirical Software Engineering*, 19(1):154–181, 2014.

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

[38] Thomas J. McCabe. A complexity measure. In *Proceedings of the 2Nd International Conference on Software Engineering*, ICSE '76, pages 407–, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[39] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. Predicting defects for eclipse. In *International Workshop on Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007.*, pages 9–9. IEEE, 2007.

[40] Adrian Schröter, Thomas Zimmermann, and Andreas Zeller. Predicting component failures at design time. In *ACM/IEEE international symposium on Empirical software engineering Proceedings of the 2006*, pages 18–27. ACM, 2006.

[41] Xi Tan, Xin Peng, Sen Pan, and Wenyun Zhao. Assessing software quality by program clustering and defect prediction. In *18th Working Conference on Reverse Engineering (WCRE), 2011*, pages 244–248. IEEE, 2011.

[42] Axel-Cyrille Ngonga Ngomo. Low-bias extraction of domain-specific concepts. *Informatica: An International Journal of Computing and Informatics*, 34(1):37–43, 2010.

[43] Ganesh J Pai and Joanne Bechta Dugan. Empirical analysis of software fault content and fault proneness using bayesian methods. *IEEE Transactions on Software Engineering*, 33(10):675–686, 2007.

[44] Yuming Zhou and Hareton Leung. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. *IEEE Transactions on Software Engineering*, 32(10):771–789, 2006.

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

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

[47] Bugzilla for mozilla. `https://bugzilla.mozilla.org/` , 2005, (accessed on April 29, 2015).

[48] Danijel Radjenović, Marjan Heričko, Richard Torkar, and Aleš Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

[49] Richard Arnold Johnson, Dean W Wichern, et al. *Applied multivariate statistical analysis*, volume 4. Prentice hall Englewood Cliffs, NJ, 1992.

[50] Jeremie Juban, Nils Siebert, and George N Kariniotakis. Probabilistic short-term wind power forecasting for the optimal management of wind generation. In *Power Tech, 2007 IEEE Lausanne*, pages 683–688. IEEE, 2007.

[51] Ekrem Kocaguneli, Tim Menzies, and Jacky W Keung. Kernel methods for software effort estimation. *Empirical Software Engineering*, 18(1):1–24, 2013.

[52] Giuseppe Scanniello and Andrian Marcus. Clustering support for static concept location in source code. In *IEEE 19th International Conference on Program Comprehension (ICPC), 2011*, pages 1–10. IEEE, 2011.

[53] Terry M Therneau. *A Package for Survival Analysis in S*, 2015. version 2.38.

[54] John Verzani. *Getting started with RStudio*. " O'Reilly Media, Inc.", 2011.

[55] Anders Hejlsberg, Scott Wiltamuth, and Peter Golde. *C# language specification*. Addison-Wesley Longman Publishing Co., Inc., 2003.

[56] Anders Hejlsberg and Mads Torgersen. Hejlsberg, anders and torgersen, mads. `Onlineathttp://msdn.microsoft.com/en-us/library/bb308966.aspx` .

[57] Apache Team. Apache repository. `http://apache.org/` , 2015, (accessed on March 29, 2015).

[58] Nachiappan Nagappan and Thomas Ball. Use of relative code churn measures to predict system defect density. In *Proceedings of 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 284–292. IEEE, 2005.

[59] Nachiappan Nagappan, Thomas Ball, and Andreas Zeller. Mining metrics to predict component failures. In *Proceedings of the 28th international conference on Software engineering*, pages 452–461. ACM, 2006.

[60] Nachiappan Nagappan and Thomas Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. In *First International Symposium on Empirical Software Engineering and Measurement, 2007. ESEM 2007.*, pages 364–373. IEEE, 2007.

[61] Thomas Zimmermann and Nachiappan Nagappan. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*, pages 531–540. ACM, 2008.

[62] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[63] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.

[64] Edmund R Malinowski. Factor analysis in chemistry. 2002.

[65] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52, 1987.

[66] Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.

[67] Jae-On Kim and Charles W Mueller. *Introduction to factor analysis: What it is and how to do it*, volume 13. Sage Beverly Hills, CA, 1978.

[68] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[69] Michel Marie Deza and Elena Deza. *Encyclopedia of Distances*. Springer Berlin Heidelberg, 2009.

[70] Evren Ceylan, F Onur Kutlubay, and Ayse Basar Bener. Software defect identification using machine learning techniques. In *32nd EUROMICRO Conference on Software Engineering and Advanced Applications, 2006. SEAA'06.*, pages 240–247. IEEE, 2006.

[71] Burak Turhan and Ayse Bener. A multivariate analysis of static code attributes for defect prediction. In *Seventh International Conference on Quality Software, 2007. QSIC'07.*, pages 231–237. IEEE, 2007.

[72] Norman Richard Draper and Harry Smith. *Applied regression analysis 2nd Ed.* New York New York John Wiley and Sons 1981., 1981.

[73] Ricardo JGB Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In *Advances in Knowledge Discovery and Data Mining*, pages 160–172. Springer, 2013.

[74] Fred R Shapiro and Michelle Pearse. Most-cited law review articles of all time, the. *Mich. L. Rev.*, 110:1483, 2011.