A Package Based Clustering Approach to Enhance the Accuracy and Performance of Software Defect Prediction

Abstract:
To enhance the accuracy and performance, software defect prediction models considering clustering of Dataset combine related and similar features to improve the learning process of the model. Here, a clustering approach named as Package Based Clustering has been proposed to group the similar and related parts of software using Object Oriented Classes' relationships and similarities. To segregate software into clusters, it performs textual analysis to identify all Object Oriented classes from source code. Then it uses package information of each class to divide those into clusters. To analyze the performance of the proposed algorithm, the linear regression model is used, which learns from clusters of related and similar classes. The experiment has been conducted on 8 releases of Xalan and Ant, and results show that the proposed technique outperforms the existing clustering algorithm that is BorderFlow and the entire system.

Keywords: Software engineering, Software testing, Software defect prediction, Package Based Clustering.

1 Introduction

Software Defect Prediction (SDP) models use software code metrics and knowledge from previous projects to predict defects for future releases of software. It can help practitioners to assess their current project status, and to reduce the software development cost by estimating the faultiness of software in advance. Many researches have been conducted using different defect prediction models, for example, Neural network, Naive Bayes, Regression modeling, Decision tree, etc. (Catal and Diri, 2009) to predict by training the model using the entire system before software testing process. Due to variabilities in software data (such as the heterogeneities among different code metrics in Dataset), prediction models considering the entire system do not always provide the desired results (Bettenburg et al., 2012). Many researches show that grouping the software source code based on their properties improves the performance of the defect prediction model (Scanniello et al., 2013; Menzies et al., 2011; Bettenburg et al., 2012). So prediction model trained from clusters of the Dataset might produce better results than those considering the entire system (Scanniello et al., 2013).

Software clustering can be accomplished by using different clustering algorithms such as BorderFlow (Scanniello et al., 2013), subtractive clustering (Sidhu et al., 2010), etc. Those clustering algorithms use software code metrics (for example, Class level, Method level, Process level, etc.) similarities or source code dependencies such as class reference to form clusters. Although those algorithms help defect prediction models improving the accuracy and performance by providing better Dataset in the training phase of the prediction model, all of those cannot always provide the best results, because of their inadequate technique (such as class references, code metrics similarities, etc.) of analyzing the source code similarities and relationships.

In recent years, several software defect prediction models have been developed. Menzies et al. (Menzies et al., 2013, 2011) 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 which only considers the code metrics and learning treatment from pairs of neighboring clusters. Scaniello et al. (Scanniello et al., 2013) proposed a defect prediction model using clustering where it considers step wise linear regression to predict defects. It uses BorderFlow algorithm to form clusters among the related classes using references between methods and attributes. Sidhu et al. (Sidhu et al., 2010) proposed a software defect prediction model which uses subtractive clustering algorithm and fuzzy inference system for early detection of defects. To predict the defects, Zimmermann et al.(Zimmermann et al., 2007) proposed the use of components for grouping software metrics, because the likelihood to fail of a component is dependent on its problem domain. All of above discussed techniques suggested to use clustering of source code in software defect prediction. So software clustering should be performed using related and similar source code chunk to improve the learning process of the prediction model as well as the prediction accuracy.

In this paper, a new technique to group the source code for software defect prediction is proposed to improve the accuracy using Package Based Clustering (PBC) rather than the entire system. PBC groups Java based software into a number of clusters using package information of each Object Oriented (OO) classes. To find clusters, PBC lists all OO classes by using textual analysis of source code. It then reads those classes to extract the package information. The clustering approach ensures that the classes having similar package information reside in the same cluster. Then the linear regression based defect prediction model is applied on those clusters. The prediction model predicts defects by establishing the relationships between software defects and a set of independent variables such as number of public methods, lack of cohesion in classes, coupling between objects, etc. In the case of small clusters when the number of observation in a cluster is smaller than the number of variables used in the defect prediction model, it combines those clusters to form a joint cluster in order to enable those for the prediction model. Now the prediction model gets better Dataset in training phase, which results better prediction accuracy.

To compare PBC with existing approaches, an experiment has been conducted on open source software which are Ant and Xalan from Promise Repository (Menzies et al., 2012). 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 to the prediction model considering BorderFlow (Scanniello et al., 2013) and the entire system. In this context, by considering the OO classes relationships and similarities, PBC outperforms the entire system in all Datasets and performs better than the BorderFlow clustering algorithm in 4 Datasets out of the total.

The paper is organized as follows. In Section 2, the methodology for improving the accuracy of software defect prediction model has been described. Section 3 is dedicated to describe the software defect prediction model, the selection of independent variables and existing clustering algorithm. Section 4 illustrates the experimental setup and result analysis of the proposed technique which also includes prediction validation approach and the selected Datasets. The related work section has been included in Section 5. And finally the conclusion is provided in Section 6.

2 Proposed Package Based Clustering Algorithm

Existing clustering techniques consider a number of source code characteristics, for example, source code dependencies or similarities, code metrics' relationships, lexical similarities to group software into multiple clusters. No such technique yet considers software project's structural information such as package information, which holds the related and similar source code together. In this paper, a new Package Based Clustering (PBC) algorithm is proposed to group software using the package information because package holds the related and similar objects in an OO system (Islam and Sakib, 2014). A Java project hierarchy to represent Project, Packages and Classes is depicted in Figure 1.

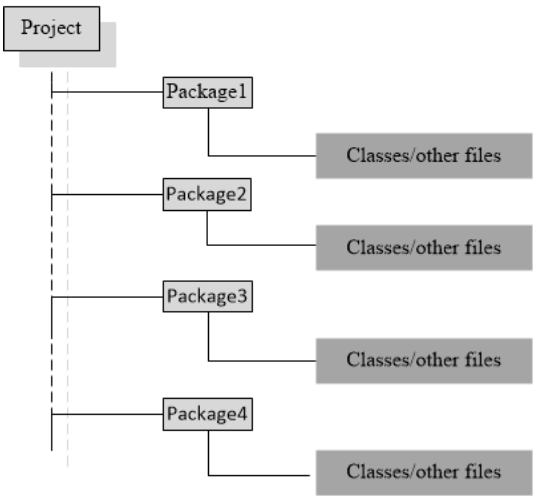

Figure 1 Software project hierarchy in Java

In Java programing convention, a package is a name space that organizes a set of related and similar classes and interfaces. Conceptually, packages are similar to different folders in a computer. A package allows a developer to group classes and interfaces together. These classes are related in some way that they might perform a specific set of tasks. Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality. As the goal of this paper is to group related and similar classes, clustering using package information (described in Algorithm 1) can easily meet the purpose.

The proposed PBC algorithm groups 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.

2.1 OO Class Identification

At the beginning of the clustering process, the proposed algorithm lists all files from software project and then it identifies potential files that will be considered for constructing clusters. In this context, software project is developed by Java, so the proposed algorithm performs OO searching by considering the file extension with $.java$. The overall OO class identification process for PBC algorithm is illustrated in the following pseudo code.

```
1: for each file do
2:   if $fileName$ ends with $.Java$ then
3:     Add file to programClassList
4:   end if
5: end for
```

In this pseudo code, it traverse all files using Line 1 and checks the file extension using Line 2. Then it only stores files having extension $.java$ using Line 3. This process stops when no files available for traversing.

2.2 Package Identification and Cluster formation

After the identification of the OO classes, PBC finds the package name of each class. To identify the 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. 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. The whole process is performed using the following psedo code.

```
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 nutshell, PBC finds 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 $6-7$.

2.3 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 paper, the SDP model was implemented using eight independent variables, (described in Section 3.2). So if the number of classes in a package is less than the number of independent 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 following lines of code.

```
1: for each package in packageContainer do
2:   if $NumberOfClassInPackage$ <$NumberOfVaribale$ then
3:     Add to joint cluster
4:   end if
5: end for
```

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 Algorithm 1.

Algorithm 1 Package based Clustering (PBC)

```
Require: Package Container $\mathcal{C}$ = 0, Package Name $\mathcal{P}$ = 0, Program Class List $\mathcal{L}$= 0, Number
    Of Varibale $\mathcal{V}$, File Name $\mathcal{F}$, Number Of Class In Package, $\mathcal{S}$.
 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 $\mathcal{P}$
 9:   if $\mathcal{P}$ contains in $\mathcal{C}$ then
10:     Add file to $\mathcal{P}$
11:   else
12:     Add new $\mathcal{P}$ To $\mathcal{C}$
13:   end if
14: end for
15: for each package in $\mathcal{C}$ do
16:   if $\mathcal{S} < \mathcal{V}$ then
17:     Add to a joint cluster
18:   end if
19: end for
```

3 Prediction Model along with Independent Variables and used Clustering Technique

This section describes the selected prediction model, its validation process and the independent variables used by the prediction model. It also highlights the clustering method that is used to compare results and finally it outlines the Datasets on which the experiment were performed.

3.1 Software Defect Prediction Model

Software defect prediction model predicts defects of a class by analyzing the relationships between software code metrics and software defects. In this paper, the linear regression model is selected as the prediction model to predict defects of a particular class because the relationships among code metrics are linear (Cohen and Cohen, 1983; Johnson et al., 1992).

The linear regression model uses a set of independent variables which are WMC, CBO, DIT, LCOM, LOC, NPM, RFC, NOC (see Section 3.2 for detail description) to predict the

Table 1 Selected independent variables used in the SDP model

Weighted Methods per classes (WMC)	It is the sum of all methods in a class.
Depth of Inheritance Tree (DIT)	It is the length of the longest path from a given class to the root class in the inheritance hierarchy.
Number of Children (NOC)	It simply measures the number of immediate descendants of a class.
Coupling Between Objects (CBO)	It counts the number of other classes to which a given class is coupled.
Response For Classes (RFC)	The RFC metric measures the number of different methods that can be executed when an object of that class receives a message.
Lack of Cohesion in Methods (LCOM)	It is the measure of counting the total number of methods in a class that are not related through the sharing of some of the class fields.
Number of Public Method (NPM)	The NPM metric simply counts all the methods in a class that are declared as public.
Lines of Codes (LOC)	It simply counts the total number of instructions in a class.

dependent variable which is the number of defects. The linear equation used by the linear regression model is given in Equation 1 (Cohen and Cohen, 1983).

$$Y = b_1x_1 + b_2x_2 + b_3x_3 + b_4x_4 + b_5x_5 + b_6x_6 + b_7x_7 + b_8x_8 + c \tag{1}$$

where,

- Y is the number of software 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 equal to 0.

3.2 Selected Independent Variables

SDP models relate the code metrics to defect proneness of software. Since PBC adopts some selected independent variables, which are the same as Basili et al. (Basili et al., 1996), identified eight significant object oriented code metrics, (listed in Table 1). The first six metrics of this table are popularly known as CK metrics, proposed by Chidamber and Kemerer (Chidamber and Kemerer, 1994) and the next two are well known size metrics. In this paper, those eight code metrics are considered as independent variables for predicting defects.

3.3 Existing Clustering Algorithms

For software defect prediction, many prominent clustering algorithms have been used to group software into multiple clusters. In this section, one of the existing clustering algorithms that is already used in the software defect prediction is described below. The proposed approach is compared to that clustering techniques to show its importance.

BorderFlow Algorithm : It is successfully implemented in (Scanniello and Marcus, 2011; Scanniello et al., 2013) to group the software into multiple clusters. In this context, the goal of this algorithm is to find groups of tightly coupled classes which are likely to be implemented a set of related features. It treats the whole software as a collection of nodes to represent 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.

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 edges such as dependencies from a subset to another subset using Equation 2. Then BorderFlow ratio can be measured using Equation 3.

$$\Omega(X, Y) = \sum e(c_i, c_j)|c_i \varepsilon X and c_j \varepsilon Y \tag{2}$$

$$F(x) = \frac{\Omega(b(X), X)}{\Omega(b(X), n(X))} \tag{3}$$

To find group of 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.

4 Experimental Setup and Result Analysis

In this section, the experimental setup and results analysis of the PBC along with BorderFlow clustering algorithm and the entire system are analyzed. It also includes the prediction validation and Dataset collection for the experiments. Finally, the effectiveness of the proposed PBC is measured by taking the mean, variance and standard deviation of absolute residuals.

4.1 Prediction Validation

To evaluate the quality of the defect prediction model achieved by the linear regression model, 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 (Kocaguneli et al., 2013; Scanniello et al., 2013), 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 (Scanniello et al., 2013).

To compare the proposed method with BorderFlow in terms of defect prediction accuracy, the $error$ is computed using the Equation 4 (Scanniello et al., 2013). The $error$ value shows how much better or worse the proposed dimension reduction technique is in the context of software defect prediction (Scanniello et al., 2013).

$$error = \frac{MAR(PBC) - MAR(BorderFlow)}{StDAR(BorderFlow)} \tag{4}$$

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 ratio 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 positive values indicate the BorderFlow outperforms the proposed PBC.

4.2 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 (Menzies et al., 2012). 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 (Team, 2015, (accessed on March 29, 2015) which are used by PBC and BorderFlow to find clusters from the source code. The short description of those 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 (Menzies et al., 2012; Team, 2015, (accessed on March 29, 2015).

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 (Menzies et al., 2012; Team, 2015, (accessed on March 29, 2015).

4.3 Experimental Setup

To perform the experiment, the prediction model using linear regression model, described in Section 3.1, is implemented using R Script (Therneau, 2015). This prediction model is validated using the validation approach, discussed in Section 4.1. The proposed clustering method PBC is based on the package information for the source code developed by Java, is implemented using C#. The BorderFlow clustering algorithm is implemented using Java as described in (Scanniello et al., 2013) 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.

To perform the whole experiment, the R statistical environment has been used to configure the experiment. In this paper, the widely used R simulation tool that is RStudio (Verzani, 2011) is used to simulate the whole experiment.

In this paper, similar to Juban et al. (Juban et al., 2007), 80% data is used to train the prediction model and the remaining data is used to assess the performance and accuracy of the model.

4.4 Result Analysis

This section presents the result analysis of the SDP model considering PBC, compared to BorderFlow algorithm and the entire system. The result of the SDP model using different

Table 2 The mean, median and standard deviation of AR values using BorderFlow, Entire System and 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

clustering approaches are compared by taking into account the MAR, MdAR and StDAR of the Absolute Residuals (AR). To evaluate the performance of the PBC based SDP model compared to BorderFlow based SDP model, both PBC and BorderFlow were applied to the Dataset (discussed in Section 4.2) to group into clusters. Finally, results of PBC and BorderFlow are analyzed and compared considering the ARs generated from the prediction model to evaluate the accuracy of these models.

Table 2 highlights the Comparison of MAR, MdAR and StDAR of AR values to identify which method produces smaller AR values. The detail inspection in this table shows that the AR value is minimum for PBC and BorderFlow Algorithms compared to the entire system. 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.

It is also needed to mention that inconsistencies 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 BorderFlow based SDP model 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 in the training purpose.

The comparison of mean value of PBC, BorderFlow and the entire system are graphically illustrated in Figure 2. In this figure, it is clearly visible that the mean value of PBC is small in 4 Datasets out of 8, compared to BorderFlow algorithm. And it also highlights that the mean value of PBC is smaller than the entire system in all Datasets.

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 4 (see Section 4.1 for detail description). Usually, the negative error value shows high accuracy and positive value shows low accuracy of the SDP model (Scanniello et al., 2013). The error values of PBC compared to BorderFlow and the entire system are calculated by applying the Equation 4. Table 3 summarizes all error values of PBC and BorderFlow. And in the same way, Table 4 also shows the error values by PBC and the entire system.

The error values considering the clustering technique PBC and BorderFlow are graphically summarized in Figure 3. In this figure, all points under the horizontal line represent software releases; for those the SDP model has better prediction accuracy. Since, the negative value means that the clustering methods using PBC outperform the clustering

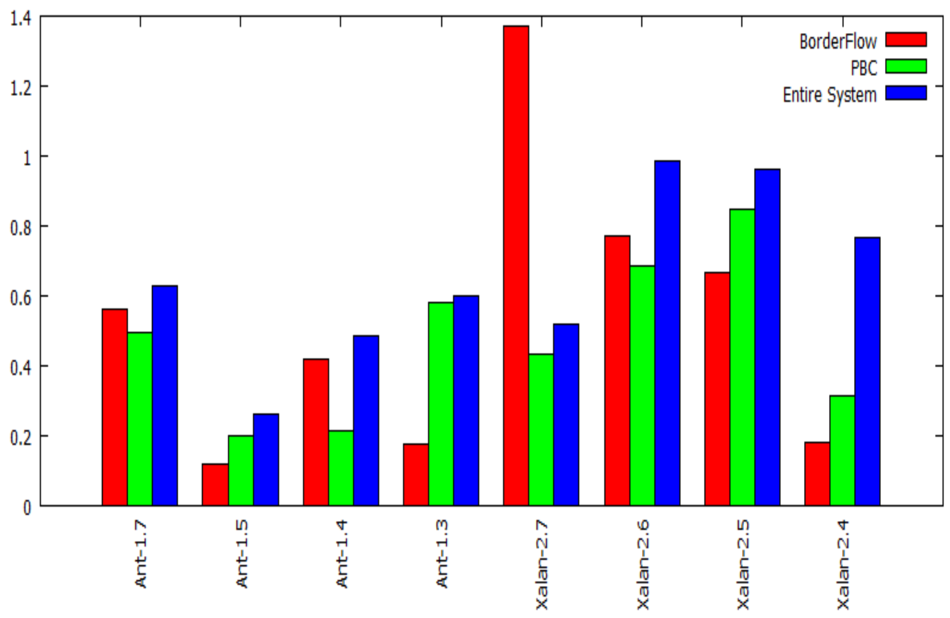

Figure 2 The AR mean values of PBC, BorderFlow and Entire System

Table 3 The error values of PBC compared to BorderFlow

Dataset	Error
Ant-1.7	-8.9%
Ant-1.5	41.6%
Ant-1.4	-50.58%
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%

Table 4 The error values of PBC compared to Entire System

Dataset	Error
Ant-1.7	-47.17%
Ant-1.5	-53.42%
Ant-1.4	-80.55%
Ant-1.3	-34.42%
Xalan-2.7	-64.61%
Xalan-2.6	-43.21%
Xalan-2.5	-22.70%
Xalan-2.4	-28.84%

methods using 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 mean 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 these Datasets because of their proper structure in the development time.

Figure 4 outlines the error values considering PBC and the entire system. Here, all software releases (such as Xalan-2.5, Xalan-2.4, Ant-1.3, etc.) reside under the horizontal line that means that SDP model has better prediction accuracy using PBC than the entire system. PBC outperforms the entire system in all Datasets because, when SDP model uses the entire system to learn, it gets some erroneous Dataset which hampers the learning process. On the other hand, when SDP model takes Dataset using PBC, it gets better Dataset

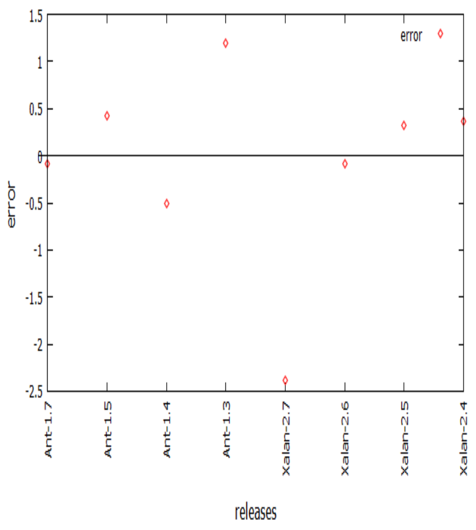

Figure 3 The distribution of error values by PBC compared to BorderFlow

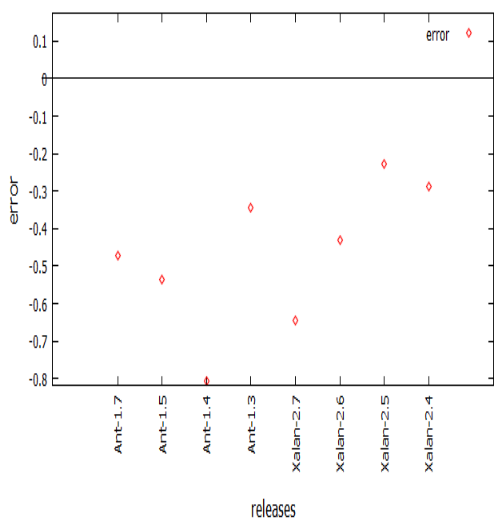

Figure 4 The distribution of error values by PBC compared to Entire System

for learning compared to the entire system. As a result, SDP model has better prediction accuracy when it considers PBC.

Concisely, it can be concluded that the PBC based SDP model performs better than the entire system because the entire system hinders the learning procedure of the SDP model. And it also outperforms BorderFlow in some software projects having proper coding structure in the development phase.

5 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 (Menzies et al., 2012) and NASA Dataset (Gray et al., 2007) 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.

5.1 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 (Schröter et al., 2006). 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 Support Vector Machine (SVM) respectively. The proposed model has been implemented on 52 eclipse plug-ins. Results show that the design and past failure history of 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 (Zimmermann et al., 2007). It uses 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 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 (Tan et al., 2011). For finding clusters, it uses Latent Semantic Indexing (LSI) to group 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 (Cohen and Cohen, 1983). 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. (Sidhu et al., 2010). 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 (Menzies et al., 2013, 2011). 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. (Bettenburg et al., 2012) 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 (Menzies et al., 2012) 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. (Scanniello et al., 2013) 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. 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 (Ngomo, 2010). 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. (Menzies et al., 2013, 2011) 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 (Scanniello et al., 2013), LSI (Tan et al., 2011), Subtractive clustering algorithm (Sidhu et al., 2010) use software code metrics, source code dependencies or code similarities, etc. to group source codes. Some approaches use PCA to reduce the dimension of the Dataset before applying the different clustering algorithms (Menzies et al., 2011, 2013; Sidhu et al., 2010). None of those methods work perfectly in all Promise Repository's Datasets (Menzies et al., 2012). So, further researches are needed to perform on clustering of source code for defect prediction.

5.2 Code Metrics Selection

The code or software metric is a quantitative measure to define the quality of software. There are lots of software metrics such as method level, class level, component level, process level, etc. (Catal and Diri, 2009). Those 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 (Catal and Diri, 2009; Pai and Bechta Dugan, 2007; Zhou and Leung, 2006). 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.

The most influential and important code metrics for OO system is CK metrics. It was proposed by Chidamber and Kemerer, to help the designers and managers by providing the inner design details of an OO system (Chidamber and Kemerer, 1994). 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 defect prediction to improve the quality of an OO system. The detail description of these metrics are given in Section 3.2.

To analyze the impact of the CK metrics (Chidamber and Kemerer, 1994) in defect prediction, Basili et al. performed an experiment using logistic regression to explore the relationship between the CK metrics and the fault-proneness of OO classes (Basili et al., 1996). 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. 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 (Gyimothy et al., 2005). 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 (Bug, 2005, (accessed on April 29, 2015). After that, it applies the statistical methods and the machine learning algorithms to the Mozilla. Results show that CBO is the best metric and WMC, RFC and LOC are the most significant metrics, while DIT and LCOM are less significant and NOC seems to be unimportant in defect prediction.

The usefulness of OO metrics such as CK metric (Chidamber and Kemerer, 1994), in defect prediction was also investigated by Zhou et al. (Zhou and Leung, 2006). It focuses on how accurately the six CK metrics can predict defects when taking the fault severity into account. Each of those 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 named as KC1. The findings of this experiment show that the CK metrics work well in low severity of defects rather than the high severity. It also shows that CBO, WMC, RFC and LCOM metrics are statistically significant in both high or low fault severity. 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 (Chidamber and Kemerer, 1994) on the number of defects and the defect proneness of a class (Pai and Bechta Dugan, 2007). It uses KC1 project from the NASA metrics data repository. It builds a Bayesian network where parent nodes are the CK metrics (Chidamber and Kemerer, 1994) and child nodes represent the fault content and fault proneness. After the model has been created, it uses Spearman correlation analysis (Cohen and Cohen, 1983) 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 papers on software defect prediction published between 1990 and 2009 (Catal and Diri, 2009). He categorized and reviewed those papers from the perspectives of metrics, learning algorithms and Datasets. It shows that the method level metrics (Halstead, 1977; McCabe, 1976) are most influential metrics in the defect prediction for Structure programing and also suggested to use class level metrics for the OO programs instead of using other metrics. Finally it recognizes that among the all class level metrics, the CK metrics is mostly used class level metrics and recommends to use this metric for OO system.

RadjenoviÂ´c et al. (Radjenović et al., 2013) 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 (Chidamber and Kemerer, 1994) metrics are the most frequently used in defect prediction. 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 (Okutan and Yıldız, 2014). For this purpose, it uses Bayesian network to determine the probabilistic influential relationships among the software code metrics. It introduces two new code metrics which are Number Of Developers (NOD) and Lack Of Coding Quality (LOCQ). An experiment has been conducted on the 9 open source Datasets from Promise Repository (Menzies et al., 2012). 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. (He et al., 2015) performed an experiment on 34 releases of 10 open source software projects to find 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.

The above discussions on the software code metrics show the class level metrics, especially CK metrics suite, is the most popular and widely used metrics in the software defect prediction. Among the CK metrics, 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. So, to incorporate all the experiments, the CK metrics along with LOC and NPM should be used as selected code metrics for the defect prediction. In this paper, the CK metrics and LOC and NPM are considered the selected code metrics for the defect prediction.

In summary, in the first section, the widely used 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 discusses the most significant works that support the selection of most influential 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.

6 Conclusion

In this paper, to perform software clustering using related and similar properties, a new clustering approach named as Package Based Clustering (PBC) is proposed. It is based on the related and similar OO classes that form packages in Java programing convention. Usually, package holds the related and similar features of software project. To group software based on the package information, it uses textual analysis on source codes to identify OO classes from software project, and then it lists those files for further analysis. To form clusters, it extracts the package information from each OO files by searching the package name. It then groups all OO files based on the package information where files having the same package belong to the same cluster, and different files from different packages reside to the different clusters. After forming clusters, it selects linear regression as the defect prediction model to predict defects by training the model using the clusters of Dataset. In special cases, if the number of OO classes in 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 selected linear regression model is conducted on Ant and Xalan considering PBC, BorderFlow and the entire system.

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 software. For BorderFlow algorithm, PBC also performs better when software package contains similar and related classes. In this context, the prediction model considering PBC outperforms the entire system in all Datasets and it also performs better than the prediction models considering BorderFlow in 4 Datasets which are Ant-1.4, Ant-1.7, Xalan-2.6, Xalan-2.7.

There is a still room for improvement in software defect prediction process by incorporating the impact of other code metrics impact to the software defects. Finally the proposed approach can be performed on some real life software projects by working collaboratively with some software companies.

References

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

V. Basili, L. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering,*, 22(10):751–761, Oct 1996. ISSN 0098-5589. doi: 10.1109/32.544352.

N. Bettenburg, M. Nagappan, and A. E. Hassan. Think locally, act globally: Improving defect and effort prediction models. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, MSR '12, pages 60–69, Piscataway, NJ, USA, 2012. IEEE Press. ISBN 978-1-4673-1761-0. URL `http://dl.acm.org/citation.cfm?id=2664446.2664455`.

C. Catal and B. Diri. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

S. Chidamber and C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, Jun 1994. ISSN 0098-5589. doi: 10.1109/32.295895.

J. Cohen and P. Cohen. *Applied multiple regression/correlation analysis for the behavioral sciences*. L. Erlbaum Associates, Hillsdale, NJ, 1983.

D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson. Nasa metric data program, June 2007. URL `http://mdp.ivv.nasa.gov/`.

T. Gyimothy, R. Ferenc, and I. Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. *Software Engineering, IEEE Transactions on*, 31(10):897–910, Oct 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.112.

M. H. Halstead. *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA, 1977. ISBN 0444002057.

P. He, B. Li, X. Liu, J. Chen, and Y. Ma. An empirical study on software defect prediction with a simplified metric set. *Inf. Softw. Technol.*, 59(C):170–190, Mar. 2015. ISSN 0950-5849. doi: 10.1016/j.infsof.2014.11.006. URL `http://dx.doi.org/10.1016/j.infsof.2014.11.006`.

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. doi: 10.1109/ICCITechn.2014.7073117.

R. A. Johnson, D. W. Wichern, et al. *Applied multivariate statistical analysis*, volume 4. Prentice hall Englewood Cliffs, NJ, 1992.

J. Juban, N. Siebert, and G. 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.

E. Kocaguneli, T. Menzies, and J. W. Keung. Kernel methods for software effort estimation. *Empirical Software Engineering*, 18(1):1–24, 2013.

T. 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. URL `http://dl.acm.org/citation.cfm?id=800253.807712`.

T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. 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.

T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, and B. Turhan. The promise repository of empirical software engineering data, June 2012. URL `http://promisedata.googlecode.com`.

T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann. Local versus global lessons for defect prediction and effort estimation. *IEEE Transactions on Software Engineering*, 39(6):822–834, 2013.

A.-C. N. Ngomo. Low-bias extraction of domain-specific concepts. *Informatica: An International Journal of Computing and Informatics*, 34(1):37–43, 2010.

A. Okutan and O. T. Yıldız. Software defect prediction using bayesian networks. *Empirical Software Engineering*, 19(1):154–181, 2014.

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

D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič. Software fault prediction metrics: A systematic literature review. *Information and Software Technology*, 55(8):1397–1418, 2013.

G. Scanniello and A. 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.

G. Scanniello, C. Gravino, A. Marcus, and T. 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.

A. Schröter, T. Zimmermann, and A. 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.

R. S. Sidhu, S. Khullar, P. S. Sandhu, R. Bedi, and K. Kaur. A subtractive clustering based approach for early prediction of fault proneness in software modules. *World Academy of Science, Engineering and Technology*, 67, 2010.

X. Tan, X. Peng, S. Pan, and W. Zhao. Assessing software quality by program clustering and defect prediction. In *18th Working Conference on Reverse Engineering (WCRE), 2011*, pages 244–248. IEEE, 2011.

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

T. M. Therneau. *A Package for Survival Analysis in S*, 2015. URL `http://CRAN.R-project.org/package=survival`. version 2.38.

J. Verzani. *Getting started with RStudio*. " O'Reilly Media, Inc.", 2011.

Y. Zhou and H. 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.

T. Zimmermann, R. Premraj, and A. 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.