Assessing Software Quality by Program Clustering and Defect Prediction

Xi Tan, Xin Peng, Sen Pan, Wenyun Zhao
School of Computer Science, Fudan University, Shanghai 201203, China
{09210240034, pengxin, 07300720171, wyzhao}@fudan.edu.cn

***Abstract*—Many empirical studies have shown that defect prediction models built on product metrics can be used to assess the quality of software modules. So far, most methods proposed in this direction predict defects by class or file. In this paper, we propose a novel software defect prediction method based on functional clusters of programs to improve the performance, especially the effort-aware performance, of defect prediction. In the method, we use proper-grained and problem-oriented program clusters as the basic units of defect prediction. To evaluate the effectiveness of the method, we conducted an experimental study on Eclipse 3.0. We found that, comparing with class-based models, cluster-based prediction models can significantly improve the recall (from 31.6% to 99.2%) and precision (from 73.8% to 91.6%) of defect prediction. According to the effort-aware evaluation, the effort needed to review code to find half of the total defects can be reduced by 6% if using cluster-based prediction models.**

***Keywords*-software quality; program clustering; defect prediction;**

I. Introduction

Under the dual pressure of time to market and cost control, software development managers often find it hard to allocate sufficient time and resources for software testing and other quality assurance (QA) activities. This often results in defect-prone products delivered to customers, and may thus cause serious problems in system dependability and safety. One cost-effective way that can alleviate this problem is to assess the quality of software modules by using defect prediction models based on software product metrics such as object-oriented (OO) design metrics [1]. Testing and QA resources thus can be allocated more efficiently according to the defect prediction results.

In this paper, we concentrate on defect prediction models based on software product metrics, including code metrics and OO design metrics. Most methods in this direction predict defects by class or file. Some work (e.g. [2], [3]) proposes to predict defects on the higher package level and gets better recall and precision. However, it is also reported [4] that package-based prediction models are less effective than class-based models when the effort is considered. The work on package-level defect prediction is based on the hypothesis that the likelihood of a component (package) to fail is significantly determined by the problem domain [2]. Furthermore, an empirical study on modern open source software [5] reveals that semantic bugs, which are application-specific and reflect inconsistence with the original design requirements or the design intention, have become the dominant root causes.

In this paper, we propose to improve software defect prediction by class-based program clustering, i.e. using program clusters rather than individual classes as the basic units of defect prediction. Program clustering usually is based on formal information (e.g. method calls and variable references) or non-formal information (e.g. identifier names and comments) [6]. The former is called structural clustering, and the latter is called semantic clustering. Intuitively, semantic clustering, which reveals the semantics and intention of the code [7], is a proper technique to identify functional clusters for defect prediction. To validate the effectiveness of our method, we conducted an empirical study on the Eclipse 3.0 system and evaluated cluster-based defect prediction models against class-based models. In particular, we are interested in the following two research questions:

- *RQ1: Are cluster-level predictions more effective than class-level predictions by both precision/recall and effort-aware evaluation?*
- *RQ2: Are structural clustering and semantic clustering of the same effectiveness for defect prediction?*

The rest of this paper is structured as follows. Section II introduces some related work, including program clustering and defect prediction. Section III describes the proposed prediction method, including clustering method, data processing and the analysis methods used. Section IV presents and evaluates the analytical results. Section V discusses related issues and analyzes the threats to validity. Finally, Section VI concludes the paper.

II. Related Work

A. Program Clustering

Program clustering groups program entities (e.g. files, classes) into modules to discover a better design or to extract significant concepts from the code [6]. Anquetil et al. [6] studied the impact of three clustering parameters on the results of program clustering, i.e. descriptive features, similarity metric and clustering algorithm. In recent years, information retrieval (IR) techniques have been widely used in program clustering and coupling measurement. The most used IR techniques in these work include Vector Space Model (VSM) and Latent Semantic Indexing (LSI). Based on LSI, Kuhn et al. [7] introduced the semantic clustering technique to group source artifacts that used similar vocabulary and interpreted the semantic clusters as linguistic topics.

B. *Software Defect Prediction*

Many empirical studies have shown that defect prediction models built on software metrics are useful for predicting software defects in source code. Among these analysis studies, linear regression and logistic regression are the ones most used in defect prediction. Linear regression is often used to predict the number of defects a program unit contains, while logistic regression is usually used to predict the likelihood that a program unit is defect-prone.

Most defect prediction methods are based on product metrics, including both code metrics and software design metrics. Some researchers also investigated the usage of development process metrics, such as change bursts [8] and communication structures [9], in defect prediction. These process metrics generally perform better than product metrics for defect prediction [4], [8]. However, the process metrics are often unavailable in new projects and projects without perfect historical records.

Although most methods in this direction predict defects by classes or files, there are also some studies that predict defects by packages or components. Kamei et al. [4] used Martin's package design metrics [10] to build package-level defect prediction models. These package- or component-level prediction models usually can achieve higher recall and precision than class-based models. However, they are usually less effective when effort is considered [4].

III. Research Method

In this section, we first give an overview of our method. Then we introduce the design of our experimental study, including the data source used and the three main steps, i.e. program clustering, data preprocessing and data analysis.

A. *Method Overview*

The process of our method is described in Figure 1. The main steps include program clustering, data preprocessing, and data analysis for defect prediction.

The first step is to extract descriptive features from source code files. And then, in program clustering, classes in source code files are grouped into functional clusters using specific clustering algorithms and similarity measurement. In the consequent data analysis, these clusters are used as basic units to build defect prediction models. Finally, a series of data analysis techniques are used to build cluster-level defect prediction models.

B. *Data Source*

In our experimental study, we used the PROMISE data set [3], which provides both pre-release and post-release defect data. We chose the post-release defect data, which reflected the number of non-trivial defects reported in the first six months after release, in our study. According to the data set, there are totally 2679 post-release defects distributed among 1568 Java files. The data set provides more than 100 types of code metrics on class level and package level [3].

To generate program clusters for defect prediction, we downloaded the source code release of Eclipse 3.0 which contains totally 10,593 Java files and roughly 1,300,000 lines of code. By analyzing the source code, we obtained a series of OO design metrics like DIT and LCOM.

TABLE I
MARTINS PACKAGE DESIGN METRICS

Metric	Description
Ca	The number of classes outside a cluster that depend on classes inside the cluster
Ce	The number of classes inside a cluster that depend on classes outside the cluster
I	$Ce/(Ca + Ce)$
NA	The number of abstract class
NC	The number of concrete class
A	$NA/(NA + NC)$
D	$\lvert I + A - 1\rvert$

C. *Clustering*

In program clustering, entity description, entity coupling (similarity metrics) and clustering algorithm are the three basic issues to be considered [6].

1) Entity description: In order to extract more abstract and high-level information from the source code, we choose the non-formal descriptive features in the clustering step (i.e. identifiers and comments of source code). More specifically, we use LSI technology to improve the initial descriptions.

2) Similarity metrics: Based on the LSI space, we calculate the similarity of two classes with the Cosine function. A higher value means a larger similarity between two classes.

3) Clustering algorithm: In the program clustering of our experiment, we use the hierarchical clustering algorithm [11], which is often used in software architecture recovery. We use the complete linkage criteria in the process, i.e. using the minimal similarity between the elements of two clusters as their similarity. As mentioned in [6], the complete linkage criteria usually results in more compact but less isolated clusters which are more cohesive. In the final hierarchy produced, each level presents a clustering result and a *cut point* should be defined to get proper-grained clusters.

D. *Data Preprocessing*

As shown in Figure 1, data preprocessing in our method is to aggregate cluster-level metrics and defect data from class-level data.

The metrics we choose for class-level defect prediction include 10 code metrics ,such as TLOC (the total lines of code) and NOM (the number of methods), from the PROMISE data set [3] and 6 OO design metrics provided by [12]. Some code metrics in the PROMISE data set have different calculation methods such as max, average and sum. We use a set of cluster-level metrics similar to the work on package-level defect prediction in [4]. We aggregate class-level code metrics to cluster-level metrics, besides, based on analysis information about class definition and dependency extracted from the source code, we calculate cluster-level design metrics according to Martin's package design metrics [13], which are presented in the Table I.

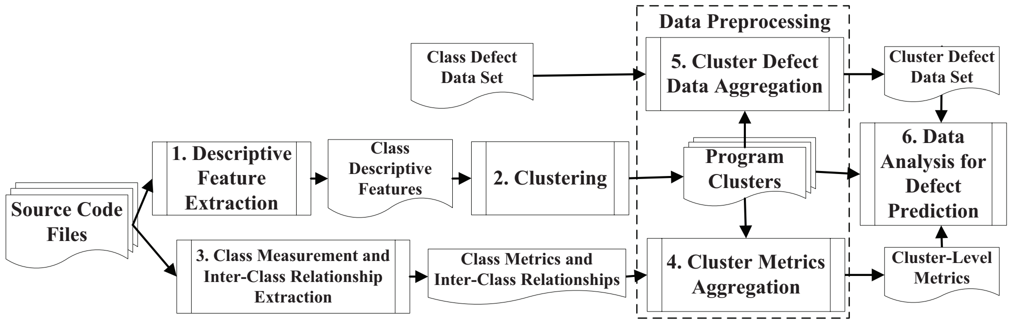
Data Preprocessing
Class Defect Data Set
5. Cluster Defect Data Aggregation
Cluster Defect Data Set
1. Descriptive Feature Extraction
Class Descriptive Features
2. Clustering
Program Clusters
6. Data Analysis for Defect Prediction
Source Code Files
3. Class Measurement and Inter-Class Relationship Extraction
Class Metrics and Inter-Class Relationships
4. Cluster Metrics Aggregation
Cluster-Level Metrics

Fig. 1. Overview of Our Method

E. Data Analysis

In this subsection, we introduce the regression analysis methods and the evaluation methods used in our study. In our experiment, linear regression and logistic regression are used to build the prediction models on both the cluster level and class level. Therefore, we use *code unit* to refer to the basic unit in defect prediction, i.e. cluster or class in our study.

In building regression models, a difficulty is the multi-collinearity problem among different product metrics of the *code units*. The multi-collinearity may lead to an inflated variance in the estimation of the metrics [14], so we use the PCA (Principal Component Analysis) technique to overcome this problem.

In order to evaluate the performance of regression models, we used *data splitting*, a commonly used evaluation technique in our experimental study. We randomly split the data set into two-thirds to build a prediction model and one-third to test the efficacy of the built model. In each experiment, the splitting evaluation is conducted 100 times to provide comprehensive and stable analysis results.

In the linear regression analysis, the prediction models are built to predict the number of defects in each *code unit*. And their prediction capability is evaluated by calculating the Pearson and Spearman correlation coefficients between the predicted number of defects and the actual number of defects provided. To evaluate how well the models explain the data set, we also check the R^2 and the adjusted R^2. R^2 is defined as the proportion of the total variation in the dependent variable that is explained by the linear regression model. Adjusted R^2 is a modification of R^2 which adjusts for the number of explanatory terms.

In the logistic regression analysis, the prediction models are built to predict the likelihood that a *code unit* may have defects. In model training, we use the 0/1 variable $S(unit)$ as dependent variable, where

$$S(unit)\begin{cases} 1, \textit{if number of defects in the unit} > 0 & \text{(1a)} \\ 0, \textit{if number of defects in the unit} = 0 & \text{(1b)} \end{cases}$$

.

The logistic regression analysis is evaluated by classification approach. We set a threshold (e.g. 0.5 in our experiment) to judge whether a *code unit* is defect-prone or not. If the likelihood of a *code unit* is larger than the threshold, it is predicted as defect-proneness, otherwise it is predicted as defect-free. Then we calculate the recall and precision as following.

- *Recall*. The recall is the percent of the defects predicted correctly in the total actual defects.
- *Precision*. The precision is the percent of the defects predicted correctly in the total predicted defects.

IV. Experimental Results

In this section, we present the results of our experimental study on the Eclipse 3.0 system in detail, including the clustering results, the linear regression results and the logistic regression results. Finally we evaluate the results of cluster-based logistic regression models against class-based models by test effort reducing.

A. Clustering Results

In the clustering process, we set the *cut point* at the level with 1,000 clusters, i.e. producing 1,000 clusters for defect prediction.

TABLE II
Clustering results

Cluster Size	Cluster Number
$2-4$	112
$5-8$	397
$9-16$	343
$17-32$	130
$33-100$	18
> 100	0
Total	1000

Table II presents the distribution of clusters with different size. It should be noted that as the largest cluster contains only 74 classes, the filtering policy actually is not used in our semantic clustering process. Clusters with less than 5 or greater than 100 elements are often regarded as extreme clusters. From Table II, we can see that about 89% of all the clusters are non-extreme, showing that by using LSI-based we have obtained quite balanced and proper-grained clusters for defect prediction.

B. Linear Regression

In our study, linear regression models are used to predict the number of defects in each *code unit*. The performance of cluster-based linear regression models is evaluated against class-based models by R^2, adjusted R^2, and Pearson and Spearman correlations. The analysis results are presented in Figure 2, in which each box plot denotes an indicator of the cluster- or class-based linear regression models over 100 runs. It can be seen that both R^2 and adjusted R^2 values of cluster-based models are much higher than class-based models. Considering that the two values are usually low in defect prediction models, the improvement achieved by cluster-based models is significant. On the other hand, since the defect data is not normally distributed, the Spearman coefficient can better reflect the correlation than Pearson coefficient. From Figure 2, it can be seen that the Spearman correlations of cluster-based models are nearly by 100% higher than class-based models on average (0.626 Vs. 0.325). The results suggest that cluster-based models can more accurately predict the number of defects than class-based models.

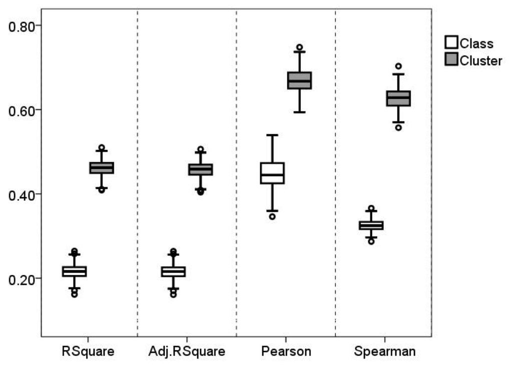

Fig. 2. Results of Linear Regression

C. Logistic Regression

In our study, logistic regression models are built to predict the likelihood that a *code unit* may have defects. The recall and precision of the classification results are calculated with the threshold 0.5. Figure 3 compares the recall and precision of cluster-based and class-based models. It can be seen that cluster-based prediction models can predict nearly all the defects (an average recall of 99.2%) with a much higher precision (91.6% on average). Thus, we can get the conclusion that cluster-based logistic regression models are more effective than class-based models in predicting defect-proneness of *code units*.

D. Test Effort Reducing

We evaluate the performance of cluster-based defect prediction models in reducing test effort against class-based models using the effort-aware evaluation model proposed in [15]. In this evaluation, we generate logistic regression models using all data in the data set and assume that testers will follow the

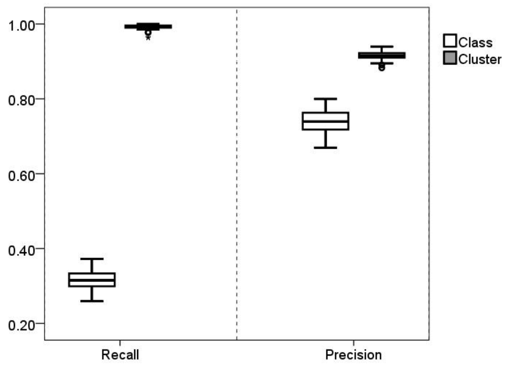

Fig. 3. Results of Logistic Regression (Classification Approach)

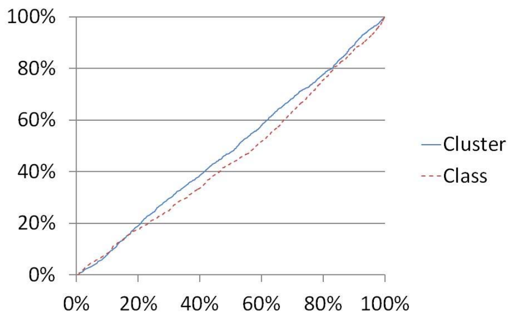

Fig. 4. Results of Testing Effort Ranking

ranking order of likelihood of defect-proneness predicted by the models to choose *code units* for review or testing.

Figure 4 presents the effort-aware evaluation of cluster-based models against class-based models. The x-axis is the percentage of lines of code to be reviewed or tested according to the defect prediction results. And the y-axis is the percentage of the defects that can be discovered according to the actual number of defects. From the comparison, we can observe that cluster-based prediction is more effective in reducing test effort. For example, to discover 50% of the total defects, testers can review or test almost 81,000 lines of code less to discover 50% of the total defects if cluster-based prediction is used. In summary, cluster-based defect prediction is more effective than class-based prediction in reducing test effort.

V. Discussion

A. Answers to the Two Research Questions

1) Research Question 1 (RQ1): RQ1 concerns the improvement of cluster-based prediction models against class-based models by both recall/precision and test effort reducing. Our answers to this question are positive when LSI-based semantic clustering is used.

From the results of our empirical study, it can be seen that the accuracy of cluster-based linear regression models is by nearly 30% points higher than class-based models. And cluster-based logistic regression models can achieve an increase of 67.6% points in recall and an increase of 17.8%

points in precision. This result is even much better than that of package-based defect prediction models.

Different from package-based prediction models, cluster-based models are even more effective than class-based models when effort is considered. According to our study on the Eclipse 3.0 system, by using cluster-based prediction testers can review or test 6% source code less to find half of the total defects. We think this improvement comes from that program clusters by semantic clustering are more proper-grained and problem-oriented than classes and packages.

2) Research Question 2 (RQ2): RQ2 concerns whether structural clustering, which uses formal descriptive features like data type references, variable accesses and method calls for clustering, is of the same effectiveness as semantic clustering for defect prediction. Intuitively, program clusters generated by semantic clustering are more problem-oriented, thus should be better choices for defect prediction. Structural clustering is based on formal features, which are at a very low abstraction level and contain much noise, thus is difficult to extract significant abstract concepts [6].

To confirm our hypothesis, we conduct a contrast study of defect prediction models using structural clustering with similar process and clustering settings. We repeat the same data splitting evaluation 100 times for linear and logistic regression analysis with structural clusters, and obtain the comparative results (average value) with prediction models based on classes and semantic clusters as shown in Table III.

From the comparative analysis, we get the answer for *RQ2* that structural clusters are not so effective as semantic clusters when used as program units for defect prediction. Actually, the results of structural clusters are even worse than class-based prediction.

TABLE III
COMPARATIVE EVALUATION OF DEFECT PREDICTION MODELS WITH STRUCTURAL CLUSTERING

	Class	Semantic Clusters	Structural Clusters
R^2	0.215	0.459	0.205
Adj. R^2	0.215	0.455	0.201
Spearman Correlation	0.325	0.626	0.319
Recall	0.316	0.992	0.378
Precision	0.738	0.916	0.750

B. Threats to Validity

One major threat is that our study is based on the data from one single project. Other projects may not have the appropriate characteristics to produce comparable results. Another threat is that we only evaluate cluster-based prediction models built with linear regression and logistic regression, other techniques should also be tested to get a comprehensive evaluation to our method. Besides, it is needed to further validate our method with more product metrics.

The other major threat lies in the great influence of clustering settings on the results of clustering. Clustering settings greatly influence the quality of clustering results. Thus the performance of cluster-based prediction models may be not so good if inappropriate clustering settings are used. And furthermore, the best clustering settings for defect prediction may vary for different systems.

Therefore, the most important significance of our study is the exploratory and indicative finding that proper-grained and problem-oriented program clusters may be a better program units for defect prediction than classes and files by both precision/recall and effort-aware evaluation.

VI. CONCLUSION

Our work in this paper is based on the hypothesis that the performance of defect prediction, especially the effort-aware performance, can be further improved by building prediction models on more proper-grained and problem-oriented program units. We propose a novel software defect prediction method based on functional clusters of programs and product metrics in this paper. To evaluate the effectiveness of the method, we conducted an empirical study on the Eclipse 3.0 system and compared the results of cluster-based defect prediction models against class-based models using linear regression analysis and logistic regression analysis. Our results confirm our hypothesis that program cluster, and specifically semantic cluster, may be a better program unit for defect prediction by both recall/reprecison and effort-aware evaluation.

REFERENCES

[1] K. Arul and H. Kohli, "Six sigma for software application of hypothesis tests to software data," *Software Quality Journal*, vol. 12, pp. 29–42, March 2004.

[2] A. Schröter, T. Zimmermann, and A. Zeller, "Predicting component failures at design time," in *ISESE*, 2006, pp. 18–27.

[3] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *PROMISE*, 2007, pp. 9–15.

[4] Y. Kamei, S. Matsumoto, A. Monden, K.-i. Matsumoto, B. Adams, and A. E. Hassan, "Revisiting common bug prediction findings using effort-aware models," in *ICSM*, 2010, pp. 1–10.

[5] Z. Li, L. Tan, X. Wang, S. Lu, Y. Zhou, and C. Zhai, "Have things changed now?: an empirical study of bug characteristics in modern open source software," in *ASID*, 2006, pp. 25–33.

[6] N. Anquetil, C. Fourrier, and T. C. Lethbridge, "Experiments with clustering as a software remodularization method," in *WCRE*, 1999, pp. 235–255.

[7] A. Kuhn, S. Ducasse, and T. Gírba, "Semantic clustering: Identifying topics in source code," *Inf. Softw. Technol.*, vol. 49, pp. 230–243, March 2007.

[8] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy, "Change bursts as defect predictors," in *ISSRE*, 2010, pp. 309–318.

[9] T. Wolf, A. Schroter, D. Damian, and T. Nguyen, "Predicting build failures using social network analysis on developer communication," in *ICSE*, 2009, pp. 1–11.

[10] R. C. Martin, *Agile Software Development, Principles, Patterns, and Practices*, 1st ed., ser. Alan Apt Series. Upper Saddle River, NJ: Prentice Hall, Oct. 2002.

[11] O. Maqbool and H. Babri, "Hierarchical clustering for software architecture recovery," *IEEE Trans. Softw. Eng.*, vol. 33, pp. 759–780, November 2007.

[12] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Trans. Softw. Eng.*, vol. 20, pp. 476–493, June 1994.

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

[14] T. Zimmermann and N. Nagappan, "Predicting defects using network analysis on dependency graphs," in *ICSE*, 2008, pp. 531–540.

[15] T. Mende and R. Koschke, "Effort-aware defect prediction models," in *CSMR*, 2010, pp. 107–116.