Journal of Theoretical and Applied Information Technology
31st December 2015. Vol.82. No.3

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

SOFTWARE DEFECT PREDICTION USING PARTICIPATION OF NODES IN SOFTWARE COUPLING

[1]MARYAM SHEKOFTEH, [2]KEYVAN MOHEBBI, [3]JAVAD KAMYABI

[1]Department Of Computer Engineering, Sarvestan Branch, Islamic Azad University, Sarvestan, Iran

[2]Department Of Electrical and Computer Engineering, Mobarakeh Branch, Islamic Azad University, Mobarakeh, Isfahan, Iran

[3] Department of E-Learning, Shiraz University, Shiraz, Iran

E-mail: [1] shekofteh@iausarv.ac.ir, [2]k.mohebbi@mau.ac.ir, [3]javadkamiabi@live.com

ABSTRACT

Testing is the best way to ensure software quality. However, this activity makes a huge challenge about limited time and financial resources. In order to achieve high quality, managers need to detect the defect prone parts of code and allocate the resources to them. Although many metrics, techniques and models have been proposed, effective identification of such parts is still a challenge. In this paper two new metrics, namely PIEDG and PIMDG are proposed for predicting defects based on dependency between the file level components of the code. The proposed metrics use the notion of participation of nodes in the software dependency graph to recognize the parts of the code which possess defects. This research also investigates the effect of ego graph on the software dependency global graph. The feasibility of the software defect predictor which is built based on the proposed metrics is evaluated. The experiments over the Eclipse bug dataset demonstrate promising results in anticipation of the software defects.

Keywords: *Defect, Prediction, Software, Metrics, Dependency Graph*

1. INTRODUCTION

In the software market, companies often face the dilemma to either deliver a software system with poor quality or lose the marketing opportunity. Both choices may have serious effects on the future of a company. Facing with defects from one release to the next may harm the image and trust of the users into the companies and postponing one release may contribute to business profit for the competitors [1].

Although software development is a troublous job and software testing can cost more than 65% of the existing resources, testing activities are often used as a "sanity checks" to minimize the risk of a defective system [1].

A big part of the job in testing activities relates to the important problem of minimizing the cost of testing activities. In the past, the important issue of assessing the cost of testing activities which should be devoted to a class was not addressed completely and the managers were left alone in strategic decision making on allocating the resources to concentrate on testing activities. For example, consider a manager who wants to promote essentially the quality of a big object-oriented system in the next release. To do so, he needs to know that which classes are vital so as to concentrate the testing activities on them and allocate its resources to them. In fact, the vital classes can be the defect-prone ones. These are classes which have the high risk of producing defect and the classes through which a defect can spread everywhere across the system. Providing a ranked list of the probable defect classes can help the manager in giving priority to testing activities based on his knowledge about the project. Therefore, the manager would get benefit from one method to recognize defective classes [1].

In this paper, we tried to evaluate the effect of node participations in software coupling. This idea evaluates the effect of each dependency in EGO view on GLOBAL view. Therefore, two metrics are proposed for predicting defect based on dependency between the components of the code. The dependency is in file level. The idea is based

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

on participation of nodes in the software dependency graph. The objective of this work is to validate the feasibility of the software defect predictor which is built based on these metrics.

This paper is organized as follows. Section two reviews the related works. Section three explains the software dependency graph. Section four proposes the new metrics. Section five includes the experimental evaluations of the proposed metrics. Finally, section six concludes the paper.

2. RELATED WORKS

Defect prediction is a very active research field and many studies have addressed this issue using a variety of different methods and techniques [2]. They mainly use the metrics and machine learning techniques to build predictive models [3, 4]. One group of these metrics recognizes the software defect through measuring code characteristics such as McCabe complexity metrics. These metrics have been proposed by some researchers and can easily be extracted from the code [5-7]. Some groups of metrics help the software teams using the dependency between the components of the code such as modules, classes and files [8, 9]. These are known as dependency metrics. The last group of metrics uses historical data or previous defects to predicate defects [10]. They follow the intuition that systems with defects in the past will also have defects in the near future [1].

Nagappan et al. [6] predict the likelihood of post-release defect by using a regression model. They researched on 5 Microsoft projects. The results of the investigation suggest that the chosen metrics can be used to predict post-release defects, successfully. However, the authors also observe that there is no single set of metrics applicable to all projects.

Ostrand et al. [10] used the information of the file status (such as new, changed and unchanged) to predict defective files. Then they used historical data from two large software systems with up to 17 releases to predict the files with the highest defect density. For each release of the two systems, the top 20% of the files with the highest predicted number of defects contained between 71% and 92% of the defects actually detected, with an overall average of 83%.

Zimmerman et al [11] reported the existing defects on Eclipse code (releases 2, 2.1 and 3) to fixes. They calculated the mapping of packages and files to the number of their defects in each considered release (in the first six months before and after release) so that they could evaluate the effective metrics in predicting the defect of software. They conducted an empirical study using common complexity metrics to build prediction models. Their models showed that a combination of complexity metrics can predict defects, suggesting that the more complex a code, the more defective. This study uses their work as a comparison basis to evaluate the newly proposed metrics.

Besides, Zimmerman & Nagappan [9] extended previous mentioned research and proposed to use network analysis on dependency graph. They showed that Network metrics derived from dependency graphs are able to predict defects in the system better than the complexity metrics and the recall for models built from network measures is by 10% points higher than for models built from complexity metrics. However, their results come from only one project (Windows Server 2003).

Turhan et al. [12] showed that defect prediction by dependency graph is much more successful than defect prediction by complexity metrics in big programs, however in short program there is not much difference.

Tan et al. [13] proposed the use of both Latent Semantic Indexing (LSI) and a hierarchical clustering algorithm to group classes that are similar at the lexical level. The authors present models to predict faulty clusters. The results suggest that the predictive models build on the clusters outperform those based on classes in terms of precision, recall, and accuracy of the faults predicted.

Menzies et al. [14], [15] used software clustering for defect predication of software. They evaluated learning for defect prediction from data local to a project or from multiple projects. Clustering the learning data is based on software metrics rather than the source code. This method is also used in [2]. The predictors obtained by combining small parts of different historical data sources (i.e., the clusters) are superior to either generalizations formed over all the data or local lessons formed from particular projects.

He et al. [16] validated the feasibility of the predictor built with a simplified metric set for software defect prediction in different scenarios. Besides, they investigated practical guidelines for the choice of training data, classifier and metric subset of a given project. The results indicated

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

that (1) the choice of training data should depend on the specific requirement of prediction accuracy; (2) the predictor built with a simplified metric set works well and is very useful in case limited resources are supplied; (3) simple classifiers (e.g., Naive Bayes) also tend to perform well when using a simplified metric set for defect prediction; and (4) in several cases, the minimum metric subset can be identified to facilitate the procedure of general defect prediction with acceptable loss of prediction precision in practice.

3. SOFTWARE DEPENDENCY GRAPH

The dependencies can be shown by construing the entire software as a low level graph in which the graph nodes are files, packages or the existing classes and the edges show the relationship or dependency between the nodes. As a result, the dependencies are proposed in file, package or class level, respectively. Such a graph is called software dependency graph [9].

In a dependency graph, the edges can be directed. The edges that exit out of a node are named as fan-out and the ones that enter in a node are named as fan-in. In this paper, each fan-out is considered as one dependency. For example, in the dependency graph of Figure 1, the fan-out of node A is 3.

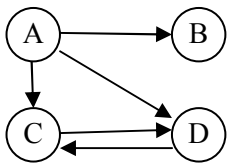

Figure1: Simple Dependency Graph

4. PROPOSING THE NEW METRICS

High coupling rate is definitely a weakness in software design. Also, fan-out is a proper gauge for coupling measurement. As a result, fan-out can be used as a measure of design quality. Consequently, numerical fan-out values can help to identify defect prone parts of code.

In Figure 1, the node A has three dependencies to the other nodes so the priority of this node is higher than that of the other nodes in order to check for possible faults.

To design the new idea, the measure networks of Ego network and Global network which are proposed in network analysis have been used. These measures have been mentioned in [9]. One important distinction made in network analysis is between ego networks and global networks.

Every node in a network has a corresponding ego network that describes how the node is connected to its neighbors. (Nodes are often referred to as "ego" in network analysis.).So an ego network for a node consists of its neighborhood in the dependency graph.

In contrast, the global network corresponds always to the entire dependency graph. While ego networks allow us to measure the local importance of a node with respect to its neighbors, global networks reveal the importance of a node within the entire software system [9].

According to the implemented studies, calculated metrics to predict the software defects are mostly related to the ego network of each node. As mentioned, Ego is the presentation of dependency graph from the view point of each node. For instance, fan-out and fan-in metrics are at Ego level because they just calculate the related dependencies of each node regardless of the whole graph. But the global view towards the dependency graph leads into proposing metrics at global level and can be useful in predicting the faults.

The new metrics in this study are proposed based on the idea of participation of nodes in the dependency graph. In other words, the participation of nodes in total coupling of dependency graph is considered to predict software defect. Unlike most of the existing metrics, this idea has a widespread view towards the graph. The proposed metrics are called PIEDG and PIMDG, and we call the set of these metrics PIDG.

The first proposed metric, namely PIEDG (Participation of each node In the Existing Dependencies in a Graph) implies that the high rate of participation in the existing software coupling causes the defect-proneness of each class.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In other words, the amount of high participation of each node in the existing dependencies in the software dependency graph makes that node defect-prone. For instance, the value of PIEDG metric for the node A in Figure 1 equals to 60%. Because the number of the existing dependencies in this graph is five of which three belongs to A.

The second proposed metric, namely PIMDG (participation of each node in the maximum possible dependencies in a graph) implies that the high rate of participation in the maximum possible software coupling can help to predict the defects. In other words, the amount of high participation of each node in the maximum possible dependencies in the software dependency graph makes that node defect-prone. In addition, the maximum rate of dependencies for n nodes in an undirected mesh graph is n(n-1). As a result, the value of PIMDG feature for node A in Figure 1 equals to 25%.

In this research, each node demonstrates one file in the eclipse project. In addition, the graph edges represent different types of dependencies between the files. The proposed metrics PIEDG and PIMDG are calculated based on the following relations.

$$PIEDG(f) = \frac{Dep(f)}{Dep(g)} \quad (1)$$

$$PIMDG(f) = \frac{Dep(f)}{n(n-1)} \quad (2)$$

Dep(f): Dependencies from f to the other nodes.

Dep(g): Number of total existing dependencies.

n: Number of nodes in the dependency graph .

After calculating these metrics, a wide variety of different classifiers can be used to test their ability in software fault prediction. The class attribute for this test is named “has-defect”. The value 1 demonstrates one or more defects that are found in the previous research works. Also the value 0 demonstrates that no fault have been found in the related class.

5. EXPERIMENTAL EVALUATIONS

To conduct the experiment, releases 2, 2.1 and 3 of Eclipse project[1] are used. In [11] Zimmerman et al. have computed the defects of the mentioned versions of this project in the file and package levels and have introduced many metrics known as Zimmerman’s metrics. The defect and metrics data of different release of this project are available online[2]. These data have been used as the basis of evaluation and comparison by this study.

In this study, to build models or predictors out of metrics and then comparison with Zimmerman metrics the Weka software version 3,6,11 has been used. To do so, different classification algorithms have been used. The class variable predicts defectiveness or non-defectiveness of each file. Our model uses the PIDG metrics and is compared with the model built out of 198 Zimmerman metrics. To test the models cross-validation method has been used.

The models or predictors built out of PIDG metrics to determine defect have been compared with the models and predictors built out of Zimmerman metrics with respect to accuracy and root mean squared error. *Accuracy* is the proximity of measurement results to the true value. Root mean squared error is a frequently used measure of the differences between values predicted by a model or a predictor and the values actually observed Tables 1, 2 and 3 show the comparative results on releases 2, 2.1 and 3 of Eclipse, respectively. These models have been built through 7 classifier algorithms.

As can be seen in Table 1, our metrics have higher accuracy than Zimmerman metrics in most cases (4 out of 7, rows 1, 2, 5 and 7) and that the improvement up to 12% has also been achieved in BaysNet classifier. With respect to the root mean squared error, our metrics in most cases (4 out of 7) show a better result and this improvement has been gotten up to 0.2 in .BaysNet

[1] http://www.eclipse.org/

[2] https://www.st.cs.uni-saarland.de/softevo/bug-data/eclipse/

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In Table 2, our metrics have been compared with Zimmerman metrics in Eclipse 2.1. As can be seen, PIDG metrics in most cases (5 out of 7) have higher accuracy than Zimmerman metrics and in some cases such as BaysNet classifier; the improvement up to 19 % has been achieved. In other cases, our metrics and Zimmerman metrics have almost the same performance. With respect to the root mean squared error our metrics in most cases (5 out of 7) show a better result and this improvement has been reached to 0.2 in BaysNet, and in the other cases our metrics and Zimmerman metrics show almost the same result.

In Table 3, the comparison has been done on Eclipse 3. As can be seen, our metrics show a better result than Zimmerman with respect to accuracy and root mean squared error in 5 cases out of 7 (all rows except 3 and 6).

In sum, the good results of the comparative evaluations with respect to the accuracy and root mean squared shows the effectiveness of our metrics in predicting software defect and it reveals that the amount of high participation of a component in existing dependencies of a graph and maximum possible dependencies of a graph causes the defect-proneness of that component. In addition, the few number of the used metrics can affect the speed of model making process.

6. CONCLUTION

In this study, PIDG metrics which includes participation in existing dependencies in a graph (PIEDG) and participation in maximum dependencies in a graph (PIMDG) metrics have been proposed. The experimental results over such metrics give an evidence on the importance of high software coupling in a weak design. This idea gives a global perspective of coupling in software dependency graph. The evaluation results also showed that PIDG produces models or predictors with higher accuracy than Zimmerman metrics in predicting the software defect and is effective for determining the defect-prone parts of the code.

REFRENCES:

[1] S. Kpodjedo, F. Ricca, P. Galinier and YG. Guéhéneuc, "design evolution metrics for defect prediction in object oriented systems", *Empirical Software Engineering, Springer*, Vol. 16, 2011, pp. 141-175.

[2] G. Scanniello, C Gravino, A. Marcus, T Menzies, "Class Level Fault Prediction Using Software Clustering", *Automated Software Engineering (ASE)*, IEEE/ACM 28th International Conference on, 2013, pp. 640- 645.

[3] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction", *Applied Soft computing, Elsevier*, Vol. 27, 2015, pp. 504-518,.

[4] Y. Suresh, L. Kumar, SK. Rath, "Statistical and machine learning methods for software fault prediction using CK metric", *Hindawi Publishing Corporation ISRN Software Engineering*, 2014, pp. 1-16.

[5] Basili VR, Briand LC, Melo VL, "A validation of object orient design metrics as quality indicators". *IEEE Transactions on Software Engineering*, 1996, pp. 75–761.

[6] N. Nagappan, T. Ball, A. Zeller, "Mining metrics to predict component failure", *In:International conference on software engineering*. China, 2006, pp. 452–461.

[7] R. Subramanyam, MS. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity implications for software defect",. *IEEE Trans Software Eng* 29:, 2003, pp. 29–310.

[8] J. Kamyabi, F. Maleki, A. Sami, "Software defect prediction using Transitive dependencies on software dependency graph", *Computer Science and Convergence*, Springer, 2012, pp. 241-249.

[9] N. Nagappan, T. Zimmermann, "Predicting defects using network analysis on dependency graph". In: *The 30th international conference on software engineering* (ICSE '08). Leipzig, Germany, 2008, pp. 531–540.

[10] T.J. Ostrand, E.J. Weyuker and R.M. Bell, "Predicting the location and number of faults in large software systems", *IEEE Transactions on Software Engineering*, Vol. 31, 2005, pp. 340-355.

[11] Thomas Zimmermann, Rahul Premraj, and Andreas Zeller. "Predicting defects for eclipse", *In Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, May 2007.

[12] A. Bener, A. Tosun, B. Turhan, "Validation of network measures as indicators of defective modules in software systems". *In: 5th International conference on predictor models in software engineering*, Vancouver, Canada, 2009.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[13] X. Tan, X. Peng, S. Pan, and W. Zhao, "Assessing software quality by program clustering and defect prediction," *in Proceedings of the Working Conference on Reverse Engineering*. IEEE Computer Society, 2011, pp. 244–248.

[14] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, and T. Zimmermann, "Local vs. global lessons for defect prediction and effort estimation," *IEEE Trans. on Softw. Eng*., no. PrePrints, 2013, pp. 1-15,.

[15] T. Menzies, A. Butcher, A. Marcus, T. Zimmermann, and D. R. Cok, "Local vs. global models for effort estimation and defect prediction," *in Proceedings of the International Conference on Automated Software Engineering*. IEEE Computer Society, 2011, pp. 343–351.

[16] P. He, B. Li, X. Liu, J. Chen, Y. Ma, "An empirical study on software defect prediction with a simplified metric set", *Information and Software Technology*, Elsevier, vol. 59, 2015, pp. 170-190.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Table1: The Comparison of The Models Built Out of PIDG Metrics and Those Built Out of Zimmerman Metrics on Eclipse 2

Measure / Classifier	Accuracy± Root mean squared error of Zimmermann's metrics	Accuracy± Root mean squared error of PIDG metrics
bayes.BayesNet	71.7046 % ± 0.526	83.638%±0.352
bayes.NaiveBayes	84.2324 % ±0.3963	85.778 %±0.3581
functions. Logistic	87.6356 % ±0.3054	86.0009 %±0.3312
meta.ClassificationViaRegression +trees.M5P	86.8777 % ±0.3166	86.0306 %±0.3257
meta.END + trees.J48	85.1241 % ±0.3732	85.6206 %±0.3325
trees.LADtree	87.2046 ±0.3113	85.9563 %±0.3573
trees.FT	83.742 % ±0.3854	85.1836 %±0.337

Table2: The Comparison of The Models Built Out of PIDG Metrics and Those Built Out of Zimmerman Metrics on Eclipse 2.1

Measure / Classifier	Accuracy± Root mean squared error of Zimmermann's metrics	Accuracy± Root mean squared error of PIDG metrics
bayes.BayesNet	70.3474% ± 0.5397	89.2748% ± 0 .3144
bayes.NaiveBayes	86.0041% ± 0.3734	88.2733% ± 0.3288
functions. Logistic	89.5157% ±0.2912	89.427% ± 0.2974
meta.ClassificationViaRegression +trees.M5P	88.1212% ± 0.3074	89.3256% ±0.2957
meta.END + trees.J48	86.4858% ± 0.3547	89.2748% ± 0.2975
trees.LADtree	89.427% ± 0.2921	89.3002% ± 0.297
trees.FT	85.2434% ± 0.3677	89.4143% ± 0.2983

Table3: The Comparison of The Models Built Out of PIDG Metrics and Those Built Out of Zimmerman Metrics on Eclipse 3

Measure / Classifier	Accuracy± Root mean squared error of Zimmermann's metrics	Accuracy± Root mean squared error of PIDG metrics
bayes.BayesNet	69.7914% ± 0.545	84.1971% ± 0.3549
bayes.NaiveBayes	83.1776% ± 0.4091	84.8107% ± 0.3693
functions. Logistic	86.6232% ± 0.3236	85.5565% ± 0.339
meta.ClassificationViaRegression +trees.M5P	84.9334% ± 0.3405	85.5565% ± 0.3333
meta.END + trees.J48	83.0265% ± 0.3979	85.4527% ± 0.336
trees.LADtree	86.3589% ± 0.3236	85.3866% ± 0.3338
trees.FT	81.6199% ± 0.4093	85.5471% ± 0.339