Volume 2, Issue 2, February 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Study of Predicting Fault Prone Software Modules

Ritika Sharma[1] *IGCE,Abhipur(Mohali)*
NehaBudhija[2] *IGCE,Abhipur(Mohali)*
Bhupinder singh[3] *IGCE,Abhipur(Mohali)*

Abstract- **Most of the current project management software's are utilizing resources on developing areas in software projects. This is considerably essential in view of the meaningful impact towards time and cost-effective development. One of the major areas is the fault proneness prediction, which is used to find out the impact areas by using several approaches, techniques and applications. Software fault proneness application is an application based on computer aided approach to predict the probability that the software contains faults. The majority of software faults are present in small number of modules, therefore accurate prediction of fault-prone modules helps to improve software quality by focusing testing efforts on a subset of modules. This paper will discuss the detail design of software fault proneness application using the object oriented approach. Prediction of fault-prone modules provides one way to support software quality engineering through improved scheduling and project control. The primary goal of our research is to develop and refine techniques for early prediction of fault-prone modules.**

Keywords: Software module, Fault-prone module, Fault detection, Software Metric, software quality.

I. Introduction

Fault-prone modules prediction is one of the most traditional and important area in Software engineering. As the complexity and the constraints under which the software is developed are increasing it is difficult to produce software without faults. Such faulty software classes may increase development and maintenance costs due to software failures, and decrease customer's satisfaction [5]. Effective prediction of fault prone software classes may enable software organizations for planning and performing testing by focusing resources on fault-prone parts of the design and code. This may result in
Significant improvement in software quality Method to identify any module that is more likely to contain a fault, from among all the modules constituting the software is called fault prone class prediction.

Detection of fault-prone modules has been widely studied [1,2,3,4].Most of these studies used some kind of software metrics, such as program complexity, size of modules, or object-oriented metrics, and constructed mathematical models to calculate fault-proneness. Early detection of fault-prone software components enables verification experts to concentrate their time and resources on the problem areas of the software system under development.

There are available metrics for predicting fault prone classes, which may help software organizations for planning and performing testing activities. This may be possible due to proper allocation of resources on fault prone parts of the design and code of the software. Hence, importance and usefulness of such metrics is understandable and important.

II. Objective

Our objective is to predict the error prone files based on metric data, so we need metric data and bug data. Source code can be obtained from source forge.net. The metric data needs to be computed based on the same method, so we used the metric tool family, Understand for C and Understand for Java. We selected the following eight metrics including C&K object oriented metrics.

- WMC (Weighted Methods per Class)
- DIT (Depth Inheritance Tree)
- NOC (Number of Children)
- CBO (Coupling Between Object Classes)
- RFC (Response for Class)
- LCOM (Lack of Cohesion Metric)

• NPM (Number of Public Methods)
• LOC (Line of Code)

III. Software Fault Proneness

Software Fault Proneness is a key factor for monitoring and controlling the quality of software. The effectiveness of analysis and testing can be easily judged by comparing the predicted distribution of fault (Fault Proneness) and amount of fault found with testing (Software faultiness).

Fault Proneness of a class predicts the probability of the presence of faults in that class. Software analysis and testing are complex and expensive activities. Estimating and preventing the faults.

Early and accurately is better approach for reducing the testing efforts. If fault prone modules are known in advance, review, analysis and testing efforts can be concentrated on those modules.

IV. Software Metrics suite uses for fault prediction

We prepared three kinds of metrics suit: history, complexity, and text filtering. We also prepared well-known classifiers for prediction methods.

A. History metrics

- FIX (Memories of bug fix) : This metric shows past existence of a fault in a module in the nominal scale. If a module had been reported bugs in the past revisions, *FIX* for the module becomes ``yes''; otherwise *FIX* is "no".

- LOCadd (Added lines of code from previous revision): This metric shows the amount of added code from the previous revision. This metric is the absolute scale.

- LOCchg (Changed lines of code from previous revision): This metric shows the amount of changed code from the previous revision. This metric is the absolute scale.

B. Complexity metrics

For the object-oriented design, metrics suit is called "CK metrics". CK metrics suit includes the following 6 metrics:

- LCOM (Lack of Cohesion on Methods): The number of pairs of member functions without shared instance variables, minus the number of pairs of functions with shared instance variables. If this subtraction is negative, the metric is set to zero.
- WMC (Weighted Methods per Class): The number of methods defined in each class.
- DIT (Depth of Inheritance Tree): The number of ancestors of a class.
- NOC (Number of Children): The number of direct descendants for each class.
- CBO (Coupling between Object classes): The number of classes to which a given class is coupled.
- RFC (Response for a Class): The number of methods that can be executed in response to a message being received by an object of that class.

C. Text filtering metrics

Pfpf *(A probability to be faulty for a module, which is calculated by a generic text filter)* [8, 9]: This metric is implicitly related to information of frequency of words in a module. The computation of *Pfpf is* rather complex, but the basic idea is simple. Assume that you have corpuses of faulty and non-faulty modules. Here, a corpus contains tokens of source code modules decomposed by the lexical analysis. When you get a new module to see whether it has a fault or not, we can determine which corpus is appropriate to contain the tokens of the new module by the Bayes theorem. This mechanism is implemented in a generic text filter. Using such a text filter, we have developed a tool to calculate *Pfpf for* a module with given corpuses of faulty and non-faulty. For our implementation, *Pfpf is* calculated by a spam filter "CRM114".

V. Conclusion

Much research on detection of fault prone software modules has been carried out so far. We have used software complexity metrics and object oriented metrics to detect fault-prone modules. We tried to confirm whether or not the metrics could improve the quality of fault-prone module detection.
The result of experiment shows that use of a certain convention of metrics can achieve higher accuracy measures.

VI.References

[1] Briand, L.C., Melo, W.L., Wust, and J.: Assessing the applicability of fault-proneness models across object oriented software projects. IEEE Trans. on Software Engineering 28(7) (2002) 706–720

[2] Khoshgoftaar, T.M., Seliya, and N.: Comparative assessment of software quality classification techniques: An empirical study. Empirical Software Engineering 9 (2004) 229–257

[3] Bellini, P., Bruno, I., Nesi, P., Rogai, D.: Comparing fault-proneness estimation models. In: Proc. of 10th IEEE International Conference on Engineering of Complex Computer Systems. (2005) 205–214

[4] Menzies, T., Greenwald, J., Frank, and A.: Data mining static code attributes to learn defect predictors. IEEE Trans.on Software Engineering 33(1) (2007) 2–13

[5] A. Koru, H. Liu, "Building effective defectprediction models in practice", *IEEE Software*, 2005, pp.23–29.

[6]K.K Aggarwal, Y. Singh, A. Kaur, R. Malhotra,"Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study", Published online in Software Process Improvement and Practice,Wiley, 2008.

[7] Mizuno, O., Kikuno, T.: Training on errors experiment to detect fault-prone software modules by spam filter.In: Proc. of 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. (2007) 405–414

[8] Mizuno, O., Kikuno, T.: Prediction of fault-prone software modules using a generic text discriminator. IEICE Trans. on Information and Systems E91-D (4) (2008) 888–896

[9] Layman, L., Kudrjavets, G., Nagappan, N.: Iterative identification of fault-prone binaries using in-process metrics. In: Proc. of 2nd International Conference on Empirical Software Engineering and Measurement. (2008)206–212

[10] D.J. Spiegelhalter, N.G. Best, B.P. Carlin, and A. van der Linde,
"Bayesian Measures of Model Complexity and Fit," J. Royal Statistical Soc., vol. 64, no. 3, pp. 583-639, 2002.

[11] T.L. Graves, A.F. Karr, J.S. Marron, and H. Siy, "Predicting Fault Incidence Using Software Change History," IEEE Trans. Software Eng., vol. 26, no. 7, pp. 653-661, July 2000.

[12] J. Gras, "End-to-End Defect Modeling," IEEE Software, vol. 21, no. 5, pp. 98-100, Sept./Oct. 2004.

[13] E.P. Minana and J. Gras, "Improving Fault Prediction Using Bayesian Networks for the Development of Embedded Software Applications," Software Testing, Verification, and Reliability, vol. 16,no. 3, pp. 157-174, 2006.

[14] H. Abdi, "Partial Least Square Regression (PLS Regression)," Encyclopedia of Measurement and Statistics, N.J. Salkind, ed. pp. 740- 744, Sage Publications, 2007.