Study of Bug Prediction Approaches

Anu Singla
Department of Computer Science and Engineering
Chandigarh University, Gharuan, Punjab, India

Vijay Kumar
Department of Computer Science and Engineering
Chandigarh University, Gharuan, Punjab, India

Abstract- **The Bug resides in the software system causes new bugs damages the software reputation and increases the cost of bug fixing effort. So it is compulsory to know or predict the existence of bug as soon as possible and remove residing bugs timely. In this paper we reviewed the previously used approaches like Code Churn, Fine Grained Source Code Changes (SCC), Fine Grained Bug Severity Prediction and also try to find their limitations.**

Keywords – Code churn, Fine grained source code changes, Bug severity prediction and Sampling based approach.

I. INTRODUCTION

Bugs in software systems are major risk and these are the one that drives most of the software maintenance cost to both, the organization that develop software and the organization that uses it. In the development process the major cost to pay for occurred bug is time and also pressure on the budget to fix the raised bug. So to fix the bug and maintain the system's quality increases the pressure on the manpower or on resources and increase their workload. Due to the workload and limited time period the quality of system suffers. To overcome this problem, numbers of bug prediction models are introduced by researchers. So the resources can allocate to those parts of the system, are more bug prone.

Software is constructed by series of changes and each change has a risk of introducing new bugs. The occurrence of new bug depends upon the granularity of change. These changes may be at commit level, file level or at statement level. Bug prediction makes use of available resources efficiently and effectively by the project development team. The bugs that occur previously are good predictors for future bugs. The main question for bug prediction is not how bug is occurring but where it is occurring. The bug could exist in previously fixed files, in modified files, in complex files or nearby other bugs. In open source software systems, user of open source software often write a "bug report" when they find bug or come across a mistake. These reports of previously occurred bugs are used to predict the same type of bugs in the new systems.

Bug databases contain a very large quantity of information of failures of software and the reasons of occurrence of failure and how it effect the system and what cost it takes to recover from failure. This information of previous occurred bugs can easily get from software archives. In this paper they work on the properties of code like how complexity of the changed code, domain of problem and quality of development process affect the occurrence of defects [2].

Mainly the defects prediction models are built using attested data from previous projects. This historical data is used on the assumption that new projects also contain the same features and using previous data, defects can be predicted. This assumption is not valid for all new projects. Some new projects contain new features and for those projects there is no available historical data. To overcome this problem there is a sampling- based approach to software defect prediction [14].

II. RELATED WORK

Phiradet Bangcharoensap et. al. [1] gave three new approaches to predict the bugs in the source code. These approaches are: text mining approach, code mining approach, change history mining approach. They also explain that the bug reports that contained specific information are simple to use to find the bugs in the files. This approach is quite useful to reassign the bugs to fixer.

Shiv kumar shivaji et. al. [3] proposed a new approach of bug prediction. According this approach machine learning classifiers are firstly trained on the history of software and then they are used to predict if the changes made in the system are causing in the bugs. But classifiers are trained on the selected features or most important features that mainly cause the bugs in the system. Trained the classifiers for all features is a difficult and very time consuming process, So selected features give better results than all features.

Yuan Tian et. al. [4] describes the severity labels to prioritize the bugs. They introduced a new model to infer severity labels from different types of information that is available from bug reports like textual and non textual. They use duplicate bug reports to measure the relative importance of every feature to check the similarity between bug reports. These similar measures are then used to assign the severity labels to similar types of bugs. In this paper they compare their proposed approach with state-of-the-art approach.

Emanuel Giger et. al. [6] analyzed the relationship between fine grained source code changes and the number of bugs in source files. This relationship analyzed using the data from the eclipse platform. On the basis of initial correlation analysis numbers of prediction models are computed using methods of machine learning. The prediction model from this analysis gives these results that there is stronger correlation between SCC and number of bugs than code churn based on lines modified (LM) and bug prone files ranked higher than non bug prone ones with the approximately 90% probability by SCC.

N. Nagappan and T. Ball [7] introduced new approach of relative code churn measures. The relative code churn measures can predict more bugs than absolute code churn measures. In the absolute measure it directly count how much total lines are modified means deleted added. Relative code churn measure the amount of code change taking place within software over unit time. It counts the how many lines added, deleted or modified differently. Number of changed lines can be easily found from the new version or a new release of a system.

A. E. Hassan [9] explains that a complex code change has more chances to create large number of faults in the software system. He gives three models for the complexity of code changes. These models are: the basic code change model, the extended code change model and the file code change model. During a particular time period first two models calculate only one value that measures the project’s overall change complexity. FCC model measures individual source file or subsystem’s overall complexity.

Sonali Agarwal and Divya Tomar [10] proposed a Linear Twin Support Vector Machine (LSTSVM) model based on feature selection technique for defect prediction. Significant feature are selected by F-score feature selection technique. These significant features are helpful to predict defects in software modules. There is a huge difference in the performance of both classifiers, the one that is developed using subset of new selected features and the one that built on all features set. They evaluated the proposed model’s defect predicting performance and used four PROMISE datasets to perform comparative analysis.

III. Approaches

A. Code Churn

Code Churn approach mainly based on the lines modified in the software system. In this, source code files purely assumed as text files. It computes lines modified in a source code file, like number of added lines, number of lines deleted from source code and lines changed per revision. In this approach, formatting source code or updating in license header generates additional lines modified although there was no change in the source code entities. For example if we change local variable or method, result will be same in both conditions.
Limitation of Code Churn approach: The main limitation of Code churn approach is that they do not achieve good performance to differentiate files into ‘files that contains bugs’ and ‘files that don’t contain bugs’.

B. Fine Grained Source code Changes (SCC)

Fine grained source code changes introduced that SCC overcomes the limitation of code churn approach. This approach captured the semantics of code changes. Fine grained source code changes approach mainly uses three hypotheses.
H1:- SCC posses a stronger correlation with the number of bugs then lines modified.
H2:- SCC gives better results to differentiate source files into buggy and not bug prone files than LM.
H3:- SCC gives better results than LM while predicting the number of bugs in source files.

C. Fine Grained Bug Severity Prediction

In this approach previous bug reports are automatically analyzed. These are analyzed on the basis of assigned severity labels. These labels assigned on the basis of how much that bug effect the system and when it is necessary to detect that bug and remove it. In this approach five severity labels are assigned named: blocker, critical, major, minor, and trivial. In this approach for comparison with the previous data, instead of using NASA data set, the files that stored in the bugzilla are used for comparison.

The limitation of this approach is that we make assumption of the complexion of the bug on the basis of its previous occurrence but there may be that bug can create more critical then the time when it occurred in other system.

D. Reducing Features to improve Code Changes Based Bug Prediction

In this approach machine learning based classifiers are trained on the selected features of history data. After that these classifiers are used on the new systems to predict the bugs that are produced due to changes in the system. If classifiers are trained on 10% important features from the all features, then they give more satisfactory and quick result.

E. Sampling-based approach to software defect prediction

In this approach some modules are taken as samples from all modules of source code. A classification model is constructed based on sample to check the quality of sampled module. Then this classification model is used on the un-sampled modules to predict the defects. CoForest and ACoForest are two semi-supervised machine learning methods to construct defect prediction models based on samples.

IV.CONCLUSION

Fine grained bug severity prediction approach does not give satisfactory result because it doesn’t contain the all information about every feature of the bug reported. Fine grained bug prediction approach only applicable for the eclipse platform. So there is need to work on this so that it can become appropriate for other software development platforms also. This approach detects the defects only on the basis of how much lines are changed not on the basis of impact of those changes. Feature selection approach only predict the history based bugs, it cannot predict the new types of bugs that introduced in the system. If these features cannot predict the all bugs then using this approach is a waste of time. The shortcoming of sampling approach is that, large software projects consists of very large number of source code files, so to choose a sample from large data and build a effective model of bug prediction is a big research challenge.

REFERENCES

[1] Phiradet Bangcharoensap, Akinori Ihara, Yasutaka Kamei, Ken-ichi Matsumoto, “Locating Source Code to be Fixed based on Initial Bug Reports”, 4th IEEE International Workshop on Empirical Software Engineering, pp. 978-0-7695-4866-1, 2012.

[2] Thomas Zimmermann, Nachiappan Nagappan, and Andreas Zeller, “Predicting Bugs from History”, Springer 2008.

[3] Shivkumar Shivaji, E. James Whitehead, Ram Akella, Sunghun Kim, “Reducing Features to Improve Code Change-Based Bug Prediction”, IEEE Transaction on Software Engineering, VOL. 39, NO. 4, APRIL 2013.

[4] Yuan Tian, David Lo, and Chengnian Sun, “Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction”, 19th IEEE Working Conference on Reverse Engineering, ISSN: 10951350, 2012.

[5] Marco D’Ambros, Michele Lanza, Romain Robbes, “An Extensive Comparison of Bug Prediction Approaches”, 7th IEEE working conference on Mining Software Repositories(MSR), ISSN: 11306120, Volume-2, Issue-1, May 2010.

[6] Emanuel Giger, Martin Pinzger, Harald C. Gall, “Comparing Fine-Grained Source Code Changes And Code Churn For Bug Prediction”, 8th IEEE Working Conference on Mining Software Repositories, ISBN: 978-1-4503-0574-7, May 2011.

[7] N. Nagappan and T. Ball, “Use of relative code churn measures to predict system defect density,” in Proceedings of ICSE 2005. ACM, pp. 284–292, 2005.

[8] N. Nagappan and T. Ball, “Static analysis tools as early indicators of pre-release defect density,” in Proceedings of ICSE 2005. ACM, pp. 580–586, 2005.

[9] A. E. Hassan, “Predicting faults using the complexity of code changes,” in Proceedings of ICSE 2009, pp. 78–88, 2009.

[10] Sonali Agarwal and Divya Tomar, “A Feature Selection Based Model for Software Defect Prediction”, in Proceedings of IJAST 2014, Vol.65, pp.39-58,2014.

[11] R. Moser, W. Pedrycz, and G. Succi, “A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction,” in Proceedings of ICSE 2008, pp. 181–190, 2008.

[12] T. Gyimothy, R. Ferenc and I. Siket, “Empirical validationof object-oriented metrics on open source software for fault prediction,” IEEE Trans. Software Eng., vol. 31, no. 10, pp. 897–910, 2005.

[13] S. Kim, T. Zimmermann, J. Whitehead, and A. Zeller, “Predicting faults from cached history,” in Proceedings of ICSE 2007. IEEE CS, pp. 489–498, 2007.

[14] Ming Li , Hongyu Zhang, Rongxin Wu, Zhi-Hua Zhou, “Sample-based software defect prediction with active and semi-supervised learning”,Springer 2011.

[15] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, “Predicting the location and number of faults in large software systems,”IEEE Trans. Software Eng., vol. 31, no. 4, pp. 340–355, 2005.

[16] A. E. Hassan and R. C. Holt, “The top ten list: Dynamic fault prediction,” in Proceedings of ICSM 2005, pp. 263–272, 2005.

[17] N. E. Fenton and N. Ohlsson, “Quantitative analysis of faults and failures in a complex software system,” IEEE Trans. Software Eng., vol. 26, no. 8, pp. 797–814, 2000.

[18] S. R. Chidamber and C. F. Kemerer, “A metrics suite for object oriented design,” IEEE Trans. Software Eng., vol. 20, no. 6, pp. 476–493, 1994.

[19] T. Zimmermann, R. Premraj, and A. Zeller, “Predicting defects for eclipse,” in Proceedings of PROMISE 2007. IEEE CS, 2007, p. 76.

[20] T. Zimmermann and N. Nagappan, “Predicting defects using network analysis on dependency graphs,” in Proceedings of ICSE 2008, 2008.

[21] A. Marcus, D. Poshyvanyk, and R. Ferenc, “Using the conceptual cohesion of classes for fault prediction in objectoriented systems,” IEEE Trans. Software Eng., vol. 34, no. 2, pp. 287–300, 2008.

[22] S. Neuhaus, T. Zimmermann, C. Holler, and A. Zeller, “Predicting vulnerable software components,” in Proceedings of CCS 2007. ACM, pp. 529–540, 2007.

[23] R. Kollmann, P. Selonen, and E. Stroulia, “A study on the current state of the art in toolsupported UML-based static reverse engineering,” in Proceedings of WCRE 2002, pp. 22–32, 2002.

[24] S. Demeyer, S. Tichelaar, and S. Ducasse, “FAMIX 2.1 — The FAMOOS Information Exchange Model,” University of Bern, Tech. Rep., 2001.

[25] M. Fischer, M. Pinzger, and H. Gall, “Populating a release history database from version control and bug tracking systems,” in Proceedings of ICSM 2003. IEEE CS, pp. 23–32, 2003.

[26] E. J. Jackson, A Users Guide to Principal Components. John Wiley & Sons Inc., 2003.

[27] G. Antoniol, K. Ayari, M. D. Penta, F. Khomh, and Y. G. Gueheneuc, “Is it a bug or an enhancement?: a textbased approach to classify change requests,” in Proceedings of CASCON 2008. ACM, pp. 304–318, 2008.