International Journal of Advanced Trends in Computer Applications
*www.ijatca.com*

# Bug fixing technique using Naive Bayes Classifier

**[1]Neha Jund, [2]Dr.Shaily Jain**
Department of Computer Science and Engineering
Chitkara University, Himachal Pradesh, India

**Abstract:** *Bug fixes are interesting, since they not only provide the source code of a bug. It also provides source code for how the bug is fixed. BugMem is popular tool to find duplicated bugs using bug fix memories: a project-specific bug and fix knowledge base developed by analysing the history of bug fixes. The change history of a software project contains a rich collection of code changes that record previous development experience. In the repository that records a software project's change history, there are various changes where developers fix bugs (known as bug fix changes) as opposed to adding new features or re-factoring source code. Changes that fix bugs are notably interesting, since they record both the old buggy code and the new fixed code. This paper presents an approach for extracting bugs fix patterns using BugMem tool and then automatically fixing the most recurrent bugs fixes using Naive Bayes classifier. Naive Bayes classifier is the best known classifier for text mining as it does not use iterative steps and hence is fast and less time consuming. Naive Bayes classifier is simple to understand and implement yet powerful. Automatically fixing the recurrent bugs will save a lot of time in debugging the software and also will save time which is put to fix the same type of bugs again which are already fixed in previous versions. Afterwards the performance of approach is checked by using ROC (receiver optimization curve) considering the false positives and true positives and also Area, Confident interval, Standard deviation.*

**Keywords:** *Bug, Fix, Open source software, Bug finding tool, Prediction, Patterns.*

## I. INTRODUCTION

The presence of bugs is a persistent quality of human created software. This is not intentional. From the dawn of software engineering and the coining of the software crisis, the creation of bug-free software has been a goal for engineer and researcher alike. One useful starting point and whether the frequency of bug kinds is similar across multiple systems. Once this information is known, it is possible to grade the kinds of bugs from most to least common, and then focus research attention on diminishing the most common types of bug.

Source code repositories hold a wealth of information that is not only useful for managing and building source code, but also as a detailed log of how the source code has evolved during development. If a piece of the source code is re-factored, evidence of this will be in the repository. The code describing how to use the software pre and post re-factoring will exist in the repository. As bugs are fixed, the changes made to correct the problem are recorded. The challenge, then, is to develop tools and techniques to automatically extract and use this information.

It is easy for programmers to think about types of bugs that might occur, and then devise a tool to look for these bugs. However, the space of possible tools to build is large. Instead of creating solutions and looking for bugs .Program maintenance and repair is one of the most time consuming and common jobs for software projects. Finding and repairing bugs in software is essential for the software to be stable, and correcting the bugs usually requires only small changes to the code base. However, finding the bugs and seeing the correct solution is not always an obvious or easy task, even for the most insignificant of software bugs.

## II. Problem formulation

**Existing work:**
In existing work the change history of 717 open source projects is mined to extract bug-fix patterns. Manual inspection of the bugs found is done to get insights into the contexts and reasons behind those bugs. Results show that missing null checks and missing initializations are very recurrent and they believe that they can be automatically detected and fixed.

**Proposed work:**
The existing approach was improved one and as well as tested over large no of source codes, but nevertheless for making the system more automated we need to replace the human analysis phase with some machine learning technique so that when the such type of bug occurs next time they get fixed automatically. Therefore in proposed algorithm we will develop a framework for automatically fixing recurrent bugs using Naïve Bayes classifier.

## III. Objective

- Collection of data to extract recurring bugs.
- To study different types of bugs for finding recurrent bug fix patterns.
- To apply normalization, stemming and labeling on bug fixes.
- To create train set for training system and test set for testing system

## IV. Techniques and Work plan

**Phase-I**
Theoretical Activity
- Reviewing existing approaches and applications using Naive Bayes classifier.
- Reviewing the various Bug fixing tools available and selecting the best one for proposed approach

**Phase-II**
Design and Testing
- Collection of data to analyze bugs.
- To analyze data and finding recurrent bug fix patterns.
- To create train set for training system and test set for testing system.
- To apply Naïve Byes classifier and performance evaluation using ROC curve (receiver operating characteristic).

## V. Related Work

Mircea Lungu et al. [1] mine the change history of 717 open source projects to extract bug-fix patterns and also manually inspect many of the bugs found to get insights into the contexts and reasons behind those bugs. Missing null checks and missing initializations are very recurrent. They can be automatically detected and fixed.

Kim et al. [2] created a tool named BugMem that extracts bug fix rules from the history of a project and applies bug detection. This approach is smart and innovative but the rules are not "patterned" and they are instead saved in a concrete form. This leads to the saved fix rules being applicable only to code clones within the same project. Code clone tracking tools would perform definitely better by following the changes of a clone and applying it on all other clones.

Anvik et al. [5] presented a semi-automated approach to assign issue reports to developers. A machine-learning algorithm is utilized on bug reports to learn the kinds of reports each developer resolves.

Chen et al. [8] created a tool (CVS Search) that searches for fragments of source code by using CVS comments. CVS Search allows one to better search the most recent version of the code by looking at previous versions to better understand the current version.

Ostrand et al. [10] describe a tool that automatically looks at the characteristics of a software project and, utilizing historical data, anticipates which files are likely to contain a larger number of faults.

Graves et al. [11] use change histories to understand how code ages. Code is to be aged if its structure makes it unnecessarily difficult to understand or maintain. Data based on change history is more useful in predicting fault rates than metrics supported on the code, such as size.

Antoniolet al. [13] describes the different types of classifier and discussed the drawback the Naïve Bayes classifier in detail. A feature selection technique applicable to classification-based bug prediction is proposed. Technique is applied to predict bugs in software changes, and execution of Naive Bayes and Support Vector Machine (SVM) classifiers is characterized The Naive Bayes classifier greatly simplify learning by assuming that features are independent given class.

R. Koschke et al. [11] demonstrate a modal for understand the data characteristics which affect the performance of naive Bayes. Their approach uses Monte Carlo simulations that allow a systematic study of classification accuracy for several classes of randomly generated problems. They analyze the impact of the distribution entropy on the categorization error, showing that low-entropy characteristic distributions yield good performance of naive Bayes. They also demonstrate that naive Bayes works well for certain nearly functional feature dependencies, thus reaching its best execution in two opposite cases: completely independent features (as expected) and functionally dependent features (which is startling). Another startling result is that the accuracy of naïve Bayes is not directly correlated with the degree of feature dependencies

measured as the class conditional mutual information between the features.

## Conclusion

In this paper, we basically reviewed various bug fixing techniques and architecture also discussed the benefits of the research as well as their importance in such buildings.

In future work, we will implement a robust and improved bug fixing technique which will outperform all the existing techniques.

## References

[1] Osman, Haidar, Mircea Lungu, and Oscar Nierstrasz. "Mining frequent bug-fix code changes." Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. IEEE, 2014.
[2] S.Kim, K. Pan and E. E. J Whitehead, Jr., "Memories of bug fixes," in Proceedings of 14th ACM SIGSOFT International symposium on Foundations of software engineering, SIGSOFT ' 06/FSE-14,(New York, NY USA).
[3]https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Naive_Bayes_classifier.html
[4]http://in.mathworks.com/help/stats/naivebayesclass.html?nocookie=true
[5] Anvik, J., Hiew, L., and Murphy, G. C., "Who Should Fix This Bug?" in Proceedings of 28th International Conference on Software Engineering (ICSE'06), Shanghai, China May 20-28 2006, pp. 361-370.
[6] http://scikit-learn.org/stable/modules/naive_bayes.html
[7]http://en.wikipedia.org/wiki/Naive_Bayes_classifier#Training.
[8] Daniel M. German" Mining CVS repositories, the softChange experience" Proceedings of the International Workshop on Software Clones (IWSC). 2009.
[9] http://www.statsoft.com/textbook/naive-bayes-classifier
[10] T.J. Ostrand, E.J. Weyuker, and R.M. Bell, "Where the Bugs Are," Proc. 2004 ACM SIGSOFT Int'l Symp. Software Testing and Analysis (ISSTA '04), July 2004.
[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] http://en.wikipedia.org/wiki/Naive_Bayes_classifier.
[13] Mockus, A., Fielding, T., and Herbsleb, D., "Two Case Studies of Open Source Software Development: Apache and Mozilla", ACM Transactions on Software Engineering and Methodology vol. 11, no. 3, July 2002, pp. 309-346.
[14]http://www.thearling.com/text/dmtechniques/dmtechniques.htm.
[15] http://en.wikipedia.org/wiki/GitHub.
[16] https://github.com/git/git.