Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X
IMPACT FACTOR: 5.258

IJCSMC, Vol. 5, Issue. 4, April 2016, pg.123 – 127

AN ANALYSIS OF VARIOUS BUG FIXING RECOMMENDATION

Mr. Karthik V[1], Mr. Siva Balan[2]

[1]M.Tech Student Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, India

[2]Senior Associate Professor,Department of Computer Science and Engineering, New Horizon College of Engineering, Bangalore, India

[1] karthikv2591@gmail.com; [2] balan.mtech@gmail.com

ABSTRACT — The most common problem faced by the programmer is that they spend lots of time in investigating files to edit and fix the bugs, as programming size increases from small to large it is important to track bugs. File recommendation is playing a very vital role, to Increase the productivity of the developers, by recommending the file to edit. Recommendation systems are one of the solutions to deal with this problem with the help of personalized services. These systems suggest reports. Recommendation has emerged as a new way for giving better recommendations. In this paper we define several metrics to understand the quality of bug reports and analyze the bug fix process, including developer involvement.

KEYWORDS: Bug fixing, Recommendation files, Mining Interaction's, Bug Reports, Analyzing the bugs.

I. Introduction

Programmers spend a significant amount of time in investigating files to edit. It has been shown in many articles that inspection can discover and remove much more bugs [1] Open source projects often maintain open bug repositories during development and maintenance, and the reporters often point out straightly or implicitly the reasons why bugs occur when they submit them. The comments about a bug are very valuable for developers to locate and fix the bug [9]

To assist programmers, researchers have developed history based recommendation systems following two paradigm The first group has mined software revision histories. These approaches make file – to - edit recommendations by mining association rules between files frequently edited together in the past. The second group has mined programmer interaction histories. These approaches mine association rules between methods or files that past programmers viewed.[MI]. This paper addresses this question . In this work, we evaluate MI (Mining programmer interaction histories), a recommendation approach considering both programmer edits and views. Using detailed view and edit histories to recommend files to edit produces the following benefits

- **Accurate recommendations.** Viewed files provide more context when programmers edit, allowing more accurate recommendations over approaches which consider only edits
- **Early recommendations.** Using view information allows recommendations to occur when programmers view files. Programmers can thus identify files to edit early, even before editing a single file.
- **Flexible recommendations.** When recommendations can occur based on viewed files, the recommendations change in response to programmers' navigation paths. This allows recommendations to occur even in scenarios that are not edit –heavy

II. RELATED RESEARCH WORK

Seonah Lee et.al , 2015 The Impact of View Histories on Edit Recommendations [1].In this paper In this paper, we presented the recommendation results at the file level , not the method level. Programmer interaction histories contain the records of files viewed by programmers as well as those edited, and, are thus a more informative source than software revision histories. we propose, MI *M*ining programmer *I*nteraction histories, recommends files to edit using the context provided by both the viewed files and the edited files. To recommend files to edit by utilizing the records of viewed files, MI mines interaction histories. MI mines interaction traces, finds association rules using the current context, and generates recommendations of files to edit. The essential part of the recommendation system is the *context* The context characterizes the situation of the programmer and is used as a query at the time of recommendation. MI predicts the files to edit with significantly higher recommendation accuracy than ROSE (about 63% over 35%), and makes recommendations earlier, often before developers begin editing. Bug Tracking System is important software typically have tens or hundreds or thousands of defects. Bug tracking system is use to manage, fix and prioritize these defects.

Pamela Bhattacharya et.al, 2013 An Empirical Analysis of Bug Reports and Bug Fixing in Open Source Android Apps [2]. In this paper Due to the novelty of the smartphone platform and tools, and the low barrier to entry for app distribution, apps are prone to errors, which affects user experience and requires frequent bug fixes. An essential step towards correcting this situation is understanding the nature of the bugs and bug-fixing processes associated with smartphone platforms and apps.
One primary factor that dominates the quality of bug reports is how well the bug has been described by the bug reporter when the report was filed A high percentage of new bugs in an application is considered a negative indicator of the application's maintenance process. to assess the quality of the triaging process, it measures how often new bugs are marked fixed, closed, or duplicate. We also found that the quality of security bug reports is higher compared to non-security bugs, though security bugs are fixed slower compared to other bugs.

Xin Xia et.al,2014 Automated Configuration Bug Report Prediction Using Text Mining [3]. In this paper the importance of configuration bugs has attracted various research studies, to detect, diagnose, and fix configuration bugs The natural-language description of a bug report provides information to indicate whether a bug is a configuration bug. Some terms in a bug report are good indicators to identify whether it is a configuration bug, while some other terms are noise. an automated tool which applies feature selection and classification techniques to build a statistical model from the natural-language description of historical bug reports, to predict whether a newly submitted bug report is a configuration bug or not. Developers could apply our model to automatically predict labels

of bug reports to improve their productivity. We develop a new automated tool which applies text mining technologies on the natural-language description of bug reports to train a statistical model on the historical bug reports with known labels to classify a new bug report as either a configuration bug report or a non-configuration bug report.

Shin Fujiwara et.al , Bug Report Recommendation for Code Inspection [4]. In this paper software projects such as Mozilla Firefox and Eclipse own more than ten thousand bug reports that have been reported but left unresolved. Bug report recommendation is a proposal of new scenario of code inspection with related bug reports, bug reports are consumed with the process of manual selection, assignment working on code reviewing all bug reports and source code are recorded in repositories, various information can be utilized, such as times developers commit messages and other meta-data of bug reports. To identify a relevant bug report, this paper employs the vector space model VSM to represent both bug reports and a source file and to make a relevancy ranking of bug reports based on a query vector of a given source file. bug report recommendation ranks bug reports based on the textual similarity between the initial source code and bug reports. Bug report recommendation finds relevant bug reports when a file of source code is given contrary to bug localization. To the best of our knowledge, this is the first study to propose the approach of bug report recommendation.

Deqing Wang et.al, 2010 Detect Related Bugs from Source Code Using Bug Information [5]. In this paper, we propose and implement a tool Rebug- Detector, which detects related bugs using bug information and code features. bug repository, users can submit bugs that they encounter while using software, and developers can confirm and fix the submitted bugs in the next release. One bug in the bug repository usually contains bug identifier, bug description, bug comments, bug resolution, bug fixed version and so on. In this paper, we propose and implement a novel related bug detection tool named Rebug-Detector There are three detailed challenges The first challenge is how to analyze bug information using Natural Language Processing NLP techniques. The second challenge is how to detect related bugs. From analyzing bug information, we can generally locate methods where bugs occur, but how to determine whether overridden or overloaded methods would cause related bugs is still difficult to deal with. We propose a general method to automatically extract bug features from bug information and code features from source code. First obtain all xml files of bugs from bug repository websites, and then we extract the title, description, comments, version, second use information extraction techniques to implement bug features extraction. Last, use some mining technique to extract code features from source code, which maybe cause bugs. using bug information and code features, we find various bugs that are related to bugs in bug repositories. The results demonstrate the usefulness of our proposal because developers do produce much overridden code.

Adrian Nistor et.al, 2013 Discovering, Reporting, and Fixing Performance Bugs [6]. In this paper, we study how performance bugs are discovered, reported to developers, and fixed by developers, and compare he results with those for non-performance bugs. Software performance is important to the overall success of a software project.. the definitions of expected and unexpected behaviors for performance bugs are vague compared to those of nonperformance bugs, are performance bugs less likely to be discovered through the observation of unexpected behaviors than non-performance bugs it is important for researchers and tool builders to understand the reasons behind the limited utilization of these techniques and address the relevant issues. a bug that affects program's correct behavior that requires additional patches to fix , and the initial patch may need cosmetic changes or to be back-ported to other software releases. Fixing performance bugs is about as likely to require supplementary patches as fixing non-performance bugs. shows, for the bugs that are fixed more than once, the total number of patches required for each bug. Performance bugs consistently need more patches than nonperformance bugs Unlike non-performance bugs, many performance bugs are found through code reasoning, not through direct observation of the bug's negative effects. Few performance bugs are found through profiling

III. CONCLUSION

Reviewing the above mention papers effectively, all the observations are presented completely.

The most common problem faced by the programmer is that they spend lots of time in investigating files to edit and fix the bugs, as programming size increases from small to large it is important to track bugs MI, a recommendation system extending ROSE, an existing edit based recommendation system. We then conducted a comparative simulation of ROSE and MI using the interaction histories stored in the Eclipse Bugzilla system MI fix the bugs by using viewed and edited histories so that the accuracy of finding bugs will be increased to 58-63% and the accuracy of MI is consistently higher than ROSE.

View information gathered from programmer interaction histories can help provide a more detailed context of programmer activity leading to more accurate, earlier and more flexible edit recommendations MI , which extends ROSE by additionally considering the records of viewed files We then conducted a simulated comparative controlled experiment by mining the records of files that programmers had both viewed and edited (MI) and mining the records of files that programmers had only edited (ROSE).
Performing a set of empirical analyses to understand the bug-fixing process in the Android platform and Android-based apps , that, although the apps are considered were started recently, their bug reports are of high quality. Also found that the quality of security bug reports is higher compared to non-security bugs, though security bugs are fixed slower compared to other bugs.
An automated tool which applies feature selection and classification techniques to build a statistical model from the natural-language description of historical bug reports, to predict whether a newly submitted bug report is a configuration bug or not. way to recommend to programmers a prioritized list of unresolved bug reports to accelerate bug detection in a given source file.
The major limitation of our current work is that we put an assumption that there exist at least one unresolved bug report related to the given source file. However, in an actual situation, there is no guarantee that all source files have a related bug report method to find related bugs from source code using bug information and code features. It first extracts bug features from bug information in bug repositories; and then it locates and extracts code features of bug method from source code, and calculates similarities comparing all overridden or overloaded methods to bug method

References

[1] Seonah Lee and Sungwon Kang *"The Impact of View Histories on Edit Recommendations"* (Volume:41 , Issue: 3),March 1 2015.

[2] Pamela Bhattacharya, Liudmila Ulanova , Iulian Neamtiu and Sai Charan Koduru *"An Empirical Analysis of Bug Reports and Bug Fixing in Open Source Android Apps"* 2013.

[3] Xin Xia , David Lo, Weiwei Qiu, Xingen Wang, and Bo Zhou *"Automated Configuration Bug Report Prediction Using Text Mining"* 2014.

[4] Shin Fujiwara , Hideaki Hata, Akito Monden and Kenichi Matsumoto *"Bug Report Recommendation for Code Inspection"*2015.

[5] Deqing Wang , Mengxiang Lin, Hui Zhang and Hongping Hu *"Detect Related Bugs from Source Code Using Bug Information"* 2010.

[6] Adrian Nistor, Tian Jiang and Lin Tan *"Discovering, Reporting, and Fixing Performance Bugs"* 2013.

[7] Shaohua Wang, Foutse Khomh and Ying Zou *" Improving Bug Localization using Correlations in Crash Reports"* 2013.

[8] Chakkrit Tantithamthavorn, Rattamont Teekavanich, Akinori Ihara and Ken-ichi Matsumoto *"Quickly Identify Bug Locations"* 2013.

[9] Makbule Gulcin Ozsoy , Faruk Polat "*Trust Based Recommendation Systems"* 2013.

