A validation of SonarQube issues related to real bugs based on open source software

Xiang Hou[1],Yao Lu[1],Yiang Gan[1], Mengwen Chen[1],Tao Wang[1],Gang Yin[1]

[1]National Laboratory for Parallel and Distributed Processing,

[1]National University of Defense Technology

[1]Changsha, China

nudt_houxiang@163.com, 839377654@qq.com, ganyiang@outlook.com, chenmengwen1991@126.com, taowang.2005@outlook.com, jack_nudt@163.com

Keywords: Bug Detection; Source Control System; SonarQube; Java; Git

Abstract. High reliability is an essential factor of modern software. At the same time, as software complexity is increasing day by day, bug counts and rate inevitably rises, leading to undermine software reliability. To avoid this problem, programmers always use issue-finding tools (bug detection) to discover the defects from source code in development of software. Recently, software inspection has been shown to be an effective way to speed up the process of source code verification and to move a portion of discovered defects from test to coding phase. As we know, modern software is often developed over many years. During this time, the commit metadata is becoming an important source of social characteristics. In this paper, our aim is to devise an empirical method to assess the percentage and the types of the issues found by issue finding tools are actual defects of the software.

1 Introduction

We always perform automatic static analysis on source code with different goals: improve important maintainability of code, check a standard compliance or detect possible defect. Sonarqube can find all violations of reasonable, bugs and bad software design pattern. After we use sonar to scan the source code, sonar will list all violations about the source code (we call the violations issues, that have the possibility to become a defect of the software). The issues reported by sonar are violations of correct programming patterns in coding process. So after we collected the issues from sonarqube, we can devise a method to predict the possibility that if the issue would be an actual defect.

So in this paper, we study the social characteristics of modern software development and the effect of sonarqube to understand the relation between issues that found by sonarqube and actual defects. To study the correlation between the actual defects and issues, we get top100 java projects from github which are controlled by git and use sonarqube to scan these projects to get the issues.

2 Experimental methods

The source code of modern software is always hosted by source control system. In our experiment, we got top 100 java projects from github. These projects always have hundreds of thousand commits in commit history. To study the correlation between sonar issue and defect of source code, we need to find all bug-fixing commits and their related bug introducing commits in the commit history. So we have to analyze the commit metadata of a project to locate all bug-fixing commits. In our experience, most developers indicate that a commit is to fix defect by including the keyword ”fix” in the commit message. Depend on the bug-fixing commit, we can confirm the specific line of code that is fixed and we can trace back to the bug-introducing commit, so we can mine the version control history (as implemented by the “git blame” command) to discover when the corresponding bugs were introduced. For the record, the bug-introducing commit may not introduce the real bug to the source code of software; it is possible to update the code for the third-party code or modify the bad software design pattern. After that, we perform automatic static analysis (like sonarqube in this paper) on the bug-introducing version and the bug-fixing version respectively.

Core methodology

Our methodology has three main steps:

1) Locate all bug-fixing commits and their related bug-introducing commits, then store the result into mysql.
2) Use sonar to scan projects respectively in bug-introducing version and bug-fixing version of source code, then collect sonar issues from database of sonarqube.
3) Account the type and percentage of sonar issue accompanied by the disappearance of bug.

Let me describe three steps in detail. Firstly, to detect the bug-fixing commit and its related bug-introducing commit, we collect top 100 java projects which are controlled by git from github. Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. Most programmers who use git to handle software development will contain the fix message in the commit message when they fix the bug. So we confirm the commit is a bug-fixing commit through mining the fix message in the commit message. We can search all commit messages for the keyword "fix" to find bug-fixing commit.

```
commit bd963bca14d8ec…..
Author: Nathan Marz <nathan@twitter.com>
Date:   Tue Oct 23 23:43:34 2012 -0700
fix bug in Cluster
diff --git a/src/jvm/… b/src/jvm/…
index b94bfdd..d268831 100644
a/src/jvm/backtype/storm/scheduler/Cluster.java
+++ b/src/jvm/backtype/storm/scheduler/Cluster.java
-       if (!this.supervisors.containsKey(host))
+       if (!this.hostToId.containsKey(host))
```

Figure 1. An example bug-fixing commit

There is an example of bug-fixing commit in Figure 1. We can find that a bug-fixing commit is consisted of a commit id; an author, identified by a name/email address pair; a commit message, which contains the keyword "fix"; and a diff, showing the lines this commit changed. Depend on the information of the bug-fixing commit, we can compute diffs for each version of the source code of the project. Because we can get the information about the modified code and the changed file address in the bug-fixing version of source code. We can use the "git blame" command to search the repository metadata to identify the bug-introducing commit. So we can switch to bug-fixing version of source code and get the file address and line of the code modified in the bug-fixing version. Finally, we obtain the bug-introducing message from the bug-fixing commit.

```
1faa92e4
(James Xu    2012-05-18 17:34:17 +0800  36)
if(!this.supervisors.containsKey(host))
```

Figure 2. git blame output for the bug-fixing commit

There is a plausible "git blame" output in Figure 2 which shows that who and when introduced the code in which commit.

```
commit 1faa92e45....
Author:JamesXu <xumingming64398966@gmail.com>
Date:   Fri May 18 17:34:17 2012 +0800

implement the scheduler mechanism for storm
….
if (!this.supervisors.containsKey(host)) {
```

Figure 3. Related bug-introducing commit

Figure 3 shows that we found the bug-introducing commit and then we can store the bug-fixing commit and bug-introducing commit into our database. If one bug-fixing commit is related to many bug-introducing commits, we can store all of these bug-introducing commits into our database.

The all process of bug-finding algorithm is like this:

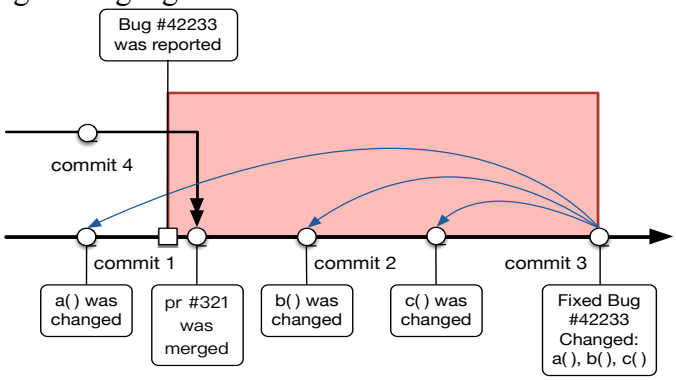

Figure 4. bug-finding process

After we got all bug-introducing commits and their related bug-fixing commits of the source code, we can use sonarqube to assess the quality of the source code respectively at the bug-introducing version and bug-fixing version.

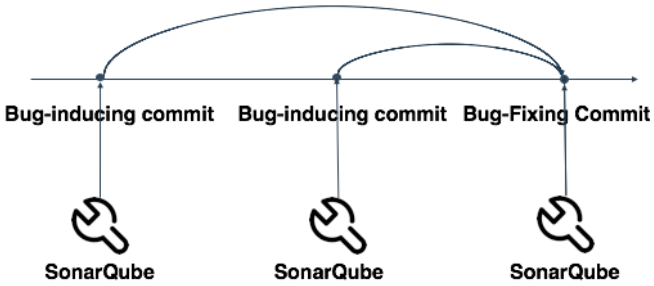

Figure 5. Process of SonarQube Accessing the Project

Finally, because the goal of our experiment is to verify which sonar issues are related to actual defects on source code and which are not, we have to find a criterion to check the relation between the sonar issue and actual defects. We believe that when one or more issues disappear in the evolution from the bug-introducing version to the bug-fixing version, and the issues were found at the same position with the bug, it's very likely these sonar issues are related to the fixed bug.

3 Results

So in this section we will present the result from carrying out by our methodology. We have performed our bug-finding algorithm on all top 100 java projects, after that it produced a large collection of defection: 112477 bug-fixing commits have been found by our bug-finding algorithm, each bug-fixing commit is related three to ten bug-introducing commits on average, then depend on these commits, we use sonarqube to scan the source code on bug-introducing version and bug-fixing version respectively to get the quality of source code, as well as we collected a large number of issues about the source code from sonarqube. We find out that 20% of all issues have co-occurrence relations with bugs that we found. These issues above who have the co-occurrence relations with bugs have a possibility to become an actual defect in the development of project. So when we use sonarqube to check the quality of source code, we need to pay attention to these issues above in case they become actual defects.

4 Conclusions and Future Work

We obtained that 4 kinds of sonarqube issues have the higher possibility to become the actual defects. The main contributions of this work are: We devise a bug-finding algorithm to mine the

social characteristics of the modern software development to confirm the bug-introducing version and bug-finding version of source code, and we provide an empirical method to validate if some sonar issue are actual defect. We can pay attention to these sonar issues to avoid the bug in development and then decrease the bug count and rate of modern software. Our future work will be devoted to open source projects written by other languages.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No.61432020 and 61472430).

References

[1] Vetro, Antonio, Maurizio Morisio, and Marco Torchiano. "An empirical validation of FindBugs issues related to defects." Evaluation & Assessment in Software Engineering (EASE 2011), 15th Annual Conference on. IET, 2011.

[2] Eyolfson, Jon, Lin Tan, and Patrick Lam. "Do time of day and developer experience affect commit bugginess?." Proceedings of the 8th Working Conference on Mining Software Repositories. ACM, 2011.

[3] Ayewah, Nathaniel, et al. "Using static analysis to find bugs." Software, IEEE25.5 (2008): 22-29.

[4] Mizuno, Osamu, and Michi Nakai. "Can faulty modules be predicted by warning messages of static code analyzer?." Advances in Software Engineering 2012 (2012): 4.

[5] Subramanyam, Ramanath, and Mayuram S. Krishnan. "Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects." Software Engineering, IEEE Transactions on 29.4 (2003): 297-310.