Understanding a Bug Introduction Change: The First Failing Commit and the Check Test.

Gema Rodríguez-Pérez
LibreSoft/GSyC
Universidad Rey Juan Carlos
Madrid, Spain
Email: gema.rodriguez@urjc.es

Jesús M. González-Barahona
LibreSoft/GSyC
Universidad Rey Juan Carlos
Madrid, Spain
Email: jgb@gsyc.es

Gregorio Robles
LibreSoft/GSyC
Universidad Rey Juan Carlos
Madrid, Spain
Email: grex@gsyc.urjc.es

***Abstract*—The assumption that "a given bug was introduced by the lines of code that were modified to fix it" seems at first glance very reasonable. In fact, many studies on bug fixing are built upon it. However, there is little empirical evidence supporting it, and a careful examination shows other possible sources for the introduction of bugs, such as an older modification, or a change in some API in a different part of the code. This paper presents an observational study designed to shed some more light in this area. For that, we studied independently the lines changed by bug fixes as a part of "the previous commit" (or commits) in two different projects. Using information from the code management, issue tracking, and code review systems, we analyzed if the code introduced by previous commits was the real bug introduction change, or on contrary it was correct at the time of introduction, meaning that there was any bug introduction change and was therefore the cause of the bug. Furthermore, we introduce a new concept, the *First Failing Commit (FFC)* to help us explaining the complex elements that take place in the bug seeding analysis. Also, we evaluate this new concept by analyzing the behavior of the implementation of a hypothetical test that is able to find the FFC as the suspicious commit to be the bug introducing change. Our results show that (surprisingly) the assumption that bugs were introduced in the previous commit does not hold for a large fraction (around 70%); the previous commits was identified as the FFC in only 30% and 35% of the analyzed commits in the two projects under study.**

I. Introduction

When a failure is found in some software, developers usually fix it by locating and modifying those source code line(s) that are the cause for the wrong behavior. It seems reasonable to assume that the immediately previous modifications of these lines are the cause of the bug. However, finding where and when a bug was introduced in the source code is not a trivial task; it may be much more complex than what this assumption suggests.

Without a way to exactly determine what line created a bug, many studies in the area of software maintenance and evolution start with the implicit assumption that the line (or lines) that is (are) being replaced in a bug fix is (are) likely the one(s) that created such bug. Below is an example of quotes from papers that use this assumption:

- bug seeding studies, e.g., *"This earlier change is the one that caused the later fixed"* [2] or *"The lines affected in the process of fixing a bug are the same one that originated or seeded that bug"* [3],
- bug fix patterns, e.g., *"The version before the bug fix revision is the bug version"* [4],
- defect prediction studies, e.g., *"A line that is deleted or changed by a bug-fixing change is a faulty line"* [5],
- tools that prevent future bugs, e.g., *"We assume that a change/commit is buggy if its modifications has been later altered by a bug-fix commit"* [6].

But although the assumption can be found frequently in the research literature, there is not enough empirical evidence supporting it. Our main goal is to determine if the previous commit that modified the same line as in the bug fix is the commit introducing the bug. With this purpose we address the following research questions:

- RQ1: When a line is changed to fix a bug, how frequently was this line the cause of a bug?
- RQ2: When a line is changed to fix a bug, how frequently is its previous change the cause of a bug?
- RQ3: How frequently can we automatically find the bug introducing change of a bug fixing?

Our first contribution is an empirical study developed manually which addresses the challenges of finding the bug introducing change in two different projects. The method focuses on the analysis of the bug fix commit at line level, looking whether the bug introducing change is the last commit that inserted the fixed line. To understand if our findings are independent of the language, we have studied two projects: `Nova`, an OpenStack subproject written in Python, and `ElasticSearch` written in Java. The second contribution is the proposal of the *First Failing Commit (FFC)* concept and the idea of a recursive test to find the bug introducing change. In order to mitigate those false positives that SZZ causes, we believe that a similar approach to the one used in the `bisection` version control system is beneficial to determine the correct bug inducing commit. In `bisection`, when an issue is resolved a test case is added. Using that test, we are able to know in which previous version the bug has been injected. Thus, going back to previous revisions we would be able to determine whether or not the bug is present. If it is indeed present, the test case will fail and the bug must have been injected before (or by) that revision. The FFC might indicate whether the commit is the real BIC or it is the first

time that the code fails after a change in other part affects the code we are testing.

Our primary findings reveal that API changes are the first reason followed by changes in the operating system, packages or requirements. As expected, we found that when only one previous change touched the lines fixed by the fix commit, we can identify it as the one that caused the bug. However, when more than one previous changes touched the fixed lines in a bug fix, in most cases only one of them is the cause of the bug and the others can not be blamed as the cause.

II. Methodology

In the case of `Nova` and `ElasticSearch`, the data needed can be obtained from the source code management, the issue tracking, and the code review systems. Figure 1 provides an overview of each step involved in our study and their outcomes.

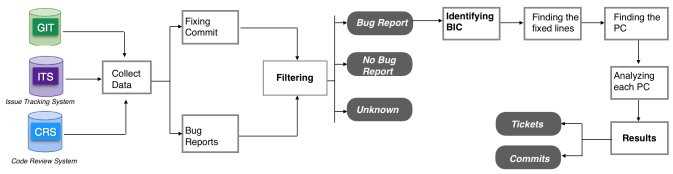

Fig. 1. Overview of the steps involved in our analysis. *PC* refers to the immediately previous commit of a fixed line

A. Identifying the Bug introducing change of a ticket

The income of this stage is a set of bug report tickets extracted randomly from the issue tracking system. All these tickets have to be closed and have a fixed commit committed and merged in the code source of the project to be able to follow the methodology described in this section.

1) Finding the lines which fixed the bug: We identify the commit that fixed the bug from the information of the bug fix commit (BFC), finding the lines that this commit added, modified or deleted and filtering out those lines that are not code.

2) For each of those lines, identify what commit added or modified or deleted last these lines: For each line touched by the BFC there is one previous commit, which we will refer to as *pc*. However, the *pc* does not have to be the same commit for all lines affected. Thus, the result of finding the *pc* of the bug is the *PC* set, which could be a set of one or more *pc*.

3) Each of these pc (and its previous commits) is analyzed to determine whether it was the bug introduced change: This analysis uses information from the BFC log and the ticket description, as well as from the log and commit changes of the *pc*.

At the end of this step we have –for each ticket analyzed– two outcomes:

- A Bug Introducing Change (BIC) does not exist: Some tickets describe a bug, but this bug was not inserted by a *pc* or some previous commit to these *pc*, therefore there is no BIC. We will refer this set of tickets as *"No BIC"*.
- A BIC exists: there is a BIC, it could be the *pc* to the changed lines by the BFC or another commit that might be in the chain of previous changes of such lines or in other part of the project. We will refer this set of tickets as *"BIC"*.

III. Evaluation

We have validated our methodology analyzing tickets from two different Open Source projects: `ElasticSearch` and `Nova`. Filtering was only necessary in the case of `Nova`, as `Launchpad` does not distinguish between bug reports and other issues. Two researchers were involved in this process, who analyzed each ticket independently. For `ElasticSearch`, we relied on its strong policy of bug labeling. We then manually analyzed tickets (that contain bugs, as a result from the filtering) to identify the BIC in both projects.

Our aim is to address some limitations of the SZZ approach by identifying the exact location of the BIC and understanding the reasons why a previous commit may not induce the fix. Hence, we introduce a new concept: the *First Failing Commit (FFC)*, which is the suspicious commit to be the bug introducing commit and ideally it is located using a test case. This concept helps to fill the scenario where lines were correct at the time of introduction but at some point a change caused that these lines become buggy, causing the bug. The FFC might be identified by using the check test idea, where a test is passed to all previous commits until it find the first that fails. We will refer to this commit as the FFC.

Figure 2 shows the check test idea. We are able to find the FFC based on the idea of having an omnipotent view. Thus, the test is passed to all previous commits looking for the one that fails. If found, we will be consider it as a candidate for the BIC.

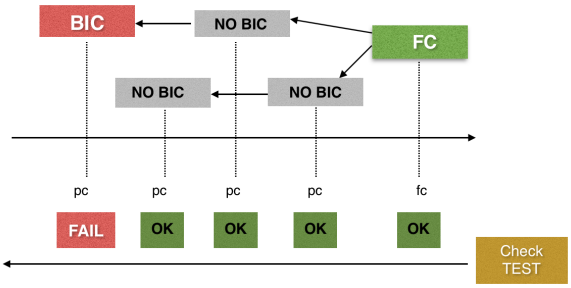

Fig. 2. Example of how we could find a candidate commit to be the bug inducing change. Each version passes or not a test written after fixing the bug in the FC (fixing commit).

A. For each of the tickets, could the hypothetical recursive test find the BIC as the FFC?

At the end of this step we have two main groups for each FFC identified:

- Group 1: The bug has been always there. Then, the FFC is the first commit in the project, and the test will always fail.

- Group 2: The FFC can be found using the test. Thus, we may have two possibilities: (a) The test fails as we expected because the failure was due to a change in other part of the code that affects the code fixed or the fixed line(s) cannot be checked before because they didn't exists. (b) The FFC is the BIC, the test fails because a previous change (the previous commit or an older commit) injected the bug.

IV. Results

We have analyzed 60 random tickets from the two projects, determining if the lines changed to fix a bug were the cause of the bug, and identifying whether a previous commit(s) is the BIC or not. Thus, after finding the lines which fix the bug in the 60 tickets in `Nova` and `ElasticSearch`, we classified the tickets into two different sets, the *"BIC"* set and the *"No BIC"* set. In addition, each previous commit also was classified into one of following sets: *"FFC"* and *"NO FFC"*.

From the 60 tickets analyzed in `Nova`, 72% were classified into BIC group and 28% into NO BIC group. The number of commits analyzed at the line level was 141 and only in 30% of the cases those commits were the FFC. Finally in 22% of the cases the FFC was the BIC.

On the contrary, from the 60 bugs in `ElasticSearch`, the number of tickets classified into set BIC was higher: 75%. However, the set of NO BIC was a little bit lower, 25%. The number of commits analyzed at line level was 132 and only in 35% of the cases those commits were the FFC. Finally in 29% of the cases, the FFC was the BIC.

Additionally, in those cases where the ticket does not present a BIC, we are able to present a short classification of the main reasons:

- Changes to APIs, such as the addition of an argument.
- Updates done, such as changes in the operating system, packages or requirements.

In any case, our research shows evidence that assuming that the pc is where the cause of a bug can be found does not hold for a significant fraction of bugs. The most common reasons for the pc was neither a FFC or a BIC in the projects are:

- Variable renaming.
- Changes done by the BFC in a clean line.
- Refactoring of the code in some lines.
- Grammar errors, dragged from former commits.

V. Conclusion

The empirical study we have performed with some bug reports from `Nova` and `ElasticSearch` has shown that for a large fraction of the analyzed tickets, around 25%, no BIC has been identified. So, in those tickets that present a BIC, the implicit assumption that bugs were introduced in the previous commit does not hold at least in 70% of them.

Our study also shows that even when we are sure that some previous change introduced the bug in the line, only in some cases the pc is the bug introducing change.

In many cases, we have identified which ones are the changes that actually introduced the bug, which could be useful to improve the accuracy of tools and models developed to prevent bugs. Also, software developers can benefit from identifying where the bug was inserted, improving their processes.

Once we have found that at least in two projects, implemented in different programming languages, the pc is in many cases not the bug introduction change, it makes sense to explore, as future work, to which extent this occurs in other projects, studying a higher number of tickets.

References

[1] ,Śliwerski, Jacek and Zimmermann, Thomas and Zeller, Andreas,When do changes induce fixes?,Proceedings of the 2005 International Workshop on Mining software repositories,1–5,2005,ACM

[2] ,Williams, Chadd and Spacco, Jaime, "Szz revisited: verifying when changes induce fixes", Proceedings of the 2008 workshop on Defects in large software systems, 32–36,2008,ACM

[3] ,Izquierdo-Cortazar, Daniel and Capiluppi, Andrea and Gonzalez-Barahona, Jesus M, Are Developers Fixing Their Own Bugs?: Tracing Bug-Fixing and Bug-Seeding Committers, International Journal of Open Source Software and Processes (IJOSSP),23–42,2011

[4] ,Pan, Kai and Kim, Sunghun and Whitehead Jr, E James,Toward an understanding of bug fix patterns,Empirical Software Engineering,286–315,2009,Springer

[5] ,Altman, Edward I,Financial ratios, discriminant analysis and the prediction of corporate bankruptcy,The journal of finance,589–609,1968,Wiley Online Library

[6] ,Fejzer, Mikołaj and Wojtyna, Michał and Burzańska, Marta and Wiśniewski, Piotr and Stencel, Krzysztof,Supporting Code Review by Automatic Detection of Potentially Buggy Changes,473–482,2015,Springer