Who introduced the bug? The importance of the previous commit

Gema Rodriguez
University King Juan Carlos
Madrid, Spain
gerope@libresoft.es

Jesus M. Gonzalez-Barahon
University King Juan Carlos
Madrid, Spain
jgb@gsyc.es

Gregorio Robles
University King Juan Carlos
Madrid, Spain
grex@gsyc.es

ABSTRACT

To fix a bug in a certain software product, some parts of its source code are modified. At first glance, it could seem reasonable that the fixed bug was introduced by the previous modification of those same parts of the source code (the previous commit). In fact, many studies on bug seeding start with this assumption. However, there is little empirical evidence supporting this assumption, and there are reasons to suppose that in some cases the bug was introduced by other actions, such as an older modification, or a change in called APIs.

This paper tries to shed some light on this area, by analyzing the relationship of bug fixes with their previous commits. To this end, we conducted an observational study on bug reports, their fixes, and their correspoonding previous commits for OpenStack. Our results show that the mentioned asumption does not hold for a large fraction of the analyzed bugs, which were not introduced by their previous commit.

Keywords

Bug introduction, bug seeding, SZZ algoritm, previous commit

1. INTRODUCTION

How to plan to apply the knowledge in a way that can benefit software engineers. What are the anticipated contributions of the work? How will you evaluate them to demonstrate usefulness?

In order to investigate if this assumption is valid, we formulate following research questions:

- RQ1 : How can tickets which are bug reports be differentiated from those that are not?
- RQ2: How often is the bug caused in the previous commit?

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MSR '16 Austin, Texas USA

© 2016 ACM. ISBN 123-4567-24-567/08/06...$15.00

DOI: 10.475/123_4

The remainder of this paper is structured as follows; first, we discuss how to extract and classify data from specific repository of OpenStack in Section **??**. In this repository we can find tickets that describe bugs; however, in order to obtain a successful results, we need know how many of them are bug reports. This problem is solved in Subsection **??**, answering in this way RQ1. After having discriminated bug reports from those that are not, to answer the RQ2 we have calculated how many times the cause of the bug could be attributed to the previous commit (see Subsection **??**). Results are then shown in Section **??**. Finally, we present a discussion, including the threats to validity, in Section **??**.

2. THE CASE OF STUDY

OpenStack was particularly of interest because of its continuously evolving due to its very active community. Although its short life, only five years, more than 5 thousand of resarchers and more than 233 thousand of commits with more than 2 Million of lines of code have contributed in the development of the project [1]. Futhermore, all history is saved and available in a version control system, being able to access to its issue tracking system [2] and the source code review [3]

OpenStack is composed by 10 projects, but we only focused in the four more actives in the 2015, Nova, Cinder, Neutron and Horizon as we can see in table 1

Table 1: Commits per Project in OpenStack

	All History	Last Year (2015)
Nova	(184) 55%	(115) 34%
Fuel	(188) 76%	(54) 22 %
Netron	(188) 56%	(116) 35%
Openstack-manuals	(184) 55%	(115) 34%
Horizon	(188) 76%	(54) 22 %
Cinder	(188) 56%	(116) 35%
Keystone	(184) 55%	(115) 34%
Heat	(188) 76%	(54) 22 %
Glance	(188) 56%	(116) 35%
Tempest	(188) 56%	(116) 35%

In this four projects we analized the relationship of bug fixes with their previous commits. In order to that, we extract a total of 459 tickets from this projects, in which we

[1] http://activity.openstack.org/dash/browser/
[2] https://launchpad.net/openstack
[3] https://review.openstack.org/

should be sure that the bug fixes come from a bug report, because two of five issues are misclassified [1] and this should cause bias in our final results.

2.1 Fist Stage: The Filtering

At this stage and with a tool's support developed to classify bug reports from other issues we distinguished the real bug reports in which afterwards analyzed the relation of the bug fixes with their previous commit.

We use the tool to analyze 459 randomly tickets from the four principal repositories in OpenStack. This tickets could be tagged as either "Fix Commited" or "Fix Released", to be able to localize the patch implemented into de source code in the version repository. They are generally tracked in Launchpad `Nova`,`Cinder`,`Horizon` and `Neutron`[4]

The parameters analyzed for each ticket, to distinguish bug reports from others, were the title and the description of the issue report and the description of the fix commit. Also, the code changes if neither the descriptions and the comments clarified the underlying ticket. Each ticket was then categorized into one of three following groups.

1. The ticket describes a bug report.
2. The ticket describes a feature, an optimization code, changes in test files or other not bug reports.
3. The ticket presents a vague description and cannot be classified without doubts.

Henceforth, we will refer to Group 1 as *Bug Report*, Group 2 as *Not Bug Report* and Group 3 as *Undecided*.

Furthermore, due to tool allows carry out a double bind analysis, three researchers, included me, used the tool to classify the 459 tickets. In 417 of the ticktes we used double bind analysis, and only those tickets classified as bug report by two of us, were considered in the next stage to analyze the relevance of their previous commits.

The table 2 show the percentage of each researcher after analyzing the tickets, and the number of tickets classifyed identically by two different researchers.

With the methodology at this stage, the researchers obtained a concordance of 70% in their classification, being able to distinguish 209 bug reports from 292 issues report using the tool.

2.1.1 The Tool

The tool was designed to solve the problem of missclassified, providing the researcher with all the relevant information needed to decide if an issue belongs to a bug report or not. The tool extracts automatically tickets from the project's repository, and offers a web-based interface which allows for collaboration, traceability and transparency of the identification of bug reports.

The image 1 shows a screenshot of anlayze tab, the principal feature of the tool where each research can comment and classify the tickets.

2.2 Second Stage: Responsability of Previous Commit

In the second stage, we only focused on the analysis of *Bug Report* group, because this group contained the tickets that we are sure were bug reports. Discarding the *Undecided* group in favour of the *Bug Report* group reduces the accuracy of the experiment due to the fact that some errors are being lost in *Undecided* group. However the *Undecided* group being only 6% of the bugs under consideration makes us think that this fact does not affect the validity of the final results.

We focused on analyzing the previous commits involved in a bug fix, understanding previous commit as the last commit that touched the line that has been fixed in the bug fix commit. The image 2 shows a real example in which the commit 31f208423 injected the lines 715, 716 and 717 in the file. But in the line 716, this commit inserted a bug. According to the description of the commmit that fixed the bug, image 3, the value in the variable of the line 716 had to be boolean, obtaining that the previous commit is responsible to cause the bug.

We analyzed a total of 209 tickets in which there were 348 previous commit, which we had to classify into one of the following groups:

1. It is responsible,
2. It is not responsible,
3. Undecided.

For that, we had to analyze the lines involved in the bug fix, in the commit parent of the bug fix commit, and to be sure that the lines was inserted/modified in the previous commit, we have to analyze these lines in the commit parent of the previous commit. This, way we can sure that the previous commit didn't copied any line that contained the bug, because in this case, the previous commit is not responsible to cause the bug.

The analisys was done manually, and we used *git blame* to see all the previos commit in each line of a involved file. In 4, we can show the steps following to discover the responsible of the bug.

1. git checkout *commmit that fix the bug*, git blame *file involved*. In this step we can see the lines added, modified or deleted by the commit that fix the bug.
2. git checkout *parent of commmit that fix the bug*, git blame *file involved*. In this step we can see all the previous commits involved in the different lines touched in the fix bug.
3. git checkout *parent of previous commit*, git blame *file involved*. In this step we can esure that the previous commit inserted these lines.

To analyze the lines of the files involved, we used therefore the `diff` command to display line-by-line difference between the file before and after the bug fix commit. `diff` is well-known algorithm, extensively explained in the research literature [3, 2], which is included in any source code management system. `diff` examines both files and returns the differences found between them, showing the line number.

At this stage, we got a list with all the previous commits, which could be where the bug was introduced. We have to identify those previous commits responsible for inserting the line with the bug. When there are more than one commit implicated in the same file is because the bug was inserted in different lines of different commits, but not everyone has

[4]https://bugs.launchpad.net/NameOfRepository

Table 2: Classification statistics of each researcher

	Bug Report	Not Bug Report	Undecided	Total
Researcher 1	(184) 55%	(115) 34%	(35) 11%	334
Researcher 2	(188) 76%	(54) 22 %	(7) 3%	249
Researcher 3	(188) 56%	(116) 35%	(30) 9%	334
Total Concordance	209	74	6	292

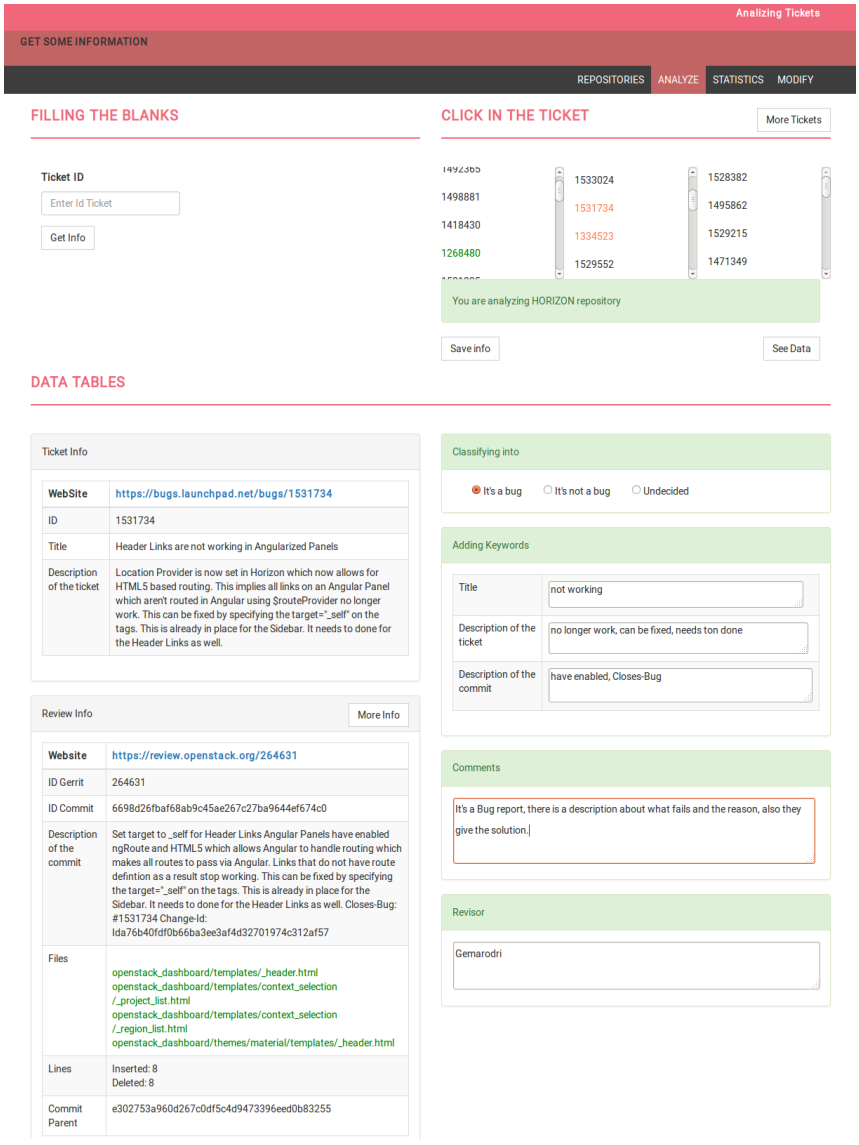

Figure 1: Screenshot of Analyze Tab

BEFORE FIX-BUG		AFTER FIX-BUG	
31f208423 711)	if rescue_auto_disk_config is None:	31f208423 711)	if rescue_auto_disk_config is None:
e30b45f69 712)	LOG.debug("auto_disk_config value not found in"	e30b45f69 712)	LOG.debug("auto_disk_config value not found in"
e30b45f69 713)	"rescue image_properties. Setting value to %s",	e30b45f69 713)	"rescue image_properties. Setting value to %s",
e30b45f69 714)	auto_disk_config, instance=instance)	e30b45f69 714)	auto_disk_config, instance=instance)
31f208423 715)	else:	31f208423 715)	else:
31f208423 716)	auto_disk_config = rescue_auto_disk_config	20847c25a 716)	auto_disk_config = strutils.bool_from_string(
		20847c25a 717)	rescue_auto_disk_config)
31f208423 717)		31f208423 718)	

Figure 2: Example of previous commit. The 31f208423 is the previous commit in the line involved in the bug-fix

```
Xen: convert image auto_disk_config value to bool before compare

During rescue mode the auto_disk_config value is pulled from the rescue
image if provided.  The value is a string but it was being used as a
boolean in an 'if' statement, leading it to be True when it shouldn't
be.  This converts it to a boolean value before comparison.

Change-Id: Ib7ffcab235ead0e770800d33c4c7cff131ca99f5
Closes-bug: 1481078
```

Figure 3: Description of the bug-fix commit

to be responsible for the commit, after the analysis the responsible can be only one of them or maybe two, in fact sometimes all of them were responsible. Therefore, comparing the status of the file, before and after the commit, we are be able to identify those previous commits that are responsible and those that are not, answering the RQ2.

After anlayzed all the previous commit involved in the 209 Bug reports, we have discovered that not all the previous commits were responsibles to cause the bug.

2.2.1 *Discarting False Positives*

We have to discart these previous commits that change the format, that add blank spaces, that change comments and that copied lines from the previous commit but.

Table 3: Probability of cause the bug depending on how many previous commits had the bug report

	After False Negative	Before False Negative
Responsible	152	152
Not responsible	154	114
Undecided	42	42

3. RESULTS

3.1 Fist Stage

We have manually analyzed 459 different tickets with support of the present tool, 125 from Cinder, 125 fron Nova, 125 from Horizon and 84 from Neutron. The table 2 show the percentage of each researcher after analyzing the tickets, 417 tickets were analyzed by two different researchers,

3.2 Second Stage

58 tickets with more than one previous commit and

Table 4: Probability of cause the bug depending on how many previous commits had the bug report

	One previous commit	More than one previous commit
Responsible	75.86%	46.32%
Not responsible	6.9%	44.21%
Undecided	17.24%	9.47%

4. DISCUSSION

This section presents first the threats to the validity of our study. Then, it discusses the possible applications and introduces the work that still has to be done.

Once we have all the tickets analyzed by diferents researchers who have used a double blind, how to proceed if there are discordances between them:

1. Should they discuss after their analysis to reach a better classification?, Should the tool provide this?

2. Does the Bug report only the same ticket classified as Bug report for all the researchers?

How to proceed if looking for the responsability of a bug when only added lines are inserted? And we are talking about a bug report not a new feature, these kinds of cases use to be when a researcher forgot check some case inside a function. [reference]

1. Is responsible the function where these lines are content?

2. Is responsible the last commit that modify something in the function?

4.1 Threats to validity

The limited sample size of tickets used in this research is the major threat to its validity.

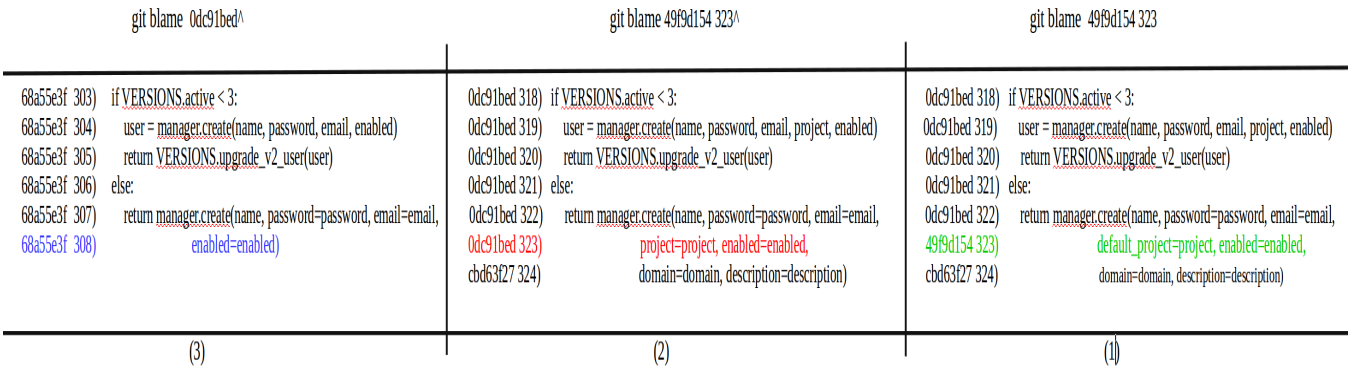

Figure 4: Process to discover the commit that caused the bug

In addition our model has threats, external and internal, that make our model not 100% valid. The internal threats are following:

- We have not taken into account errors that have been classified into *Undecided*.
- There could be some lax criteria involving the subjective opinion of the reviewers.
- We are not experts in analyzing and classifying tickets, and our inexperience may have influenced the results of the analysis.
- We are only using part of the information that the ticket provides, like comments and text. There could be a recognized pattern, unknown at first sight, that involves other parts of the information, or the whole information.

The external threats, related to the researchers that have conducted the classification, are following:

- The word *bug* is continuously mentioned in the description and commit of a ticket even when we found it is not an error. This could lead to the incorrect classification during the reviewing process.
- Some tickets are not explicitly described, which could increase the percentage of *Undecided*. This is especially true if the reviewers are not from OpenStack.

5. CONCLUSIONS AND FUTURE WORK

In the preliminary results we have observed that the current premise, based on the SZZ algorithm, does not hold in all cases. Around half of the previous commits analyzed have been identified as responsible for inserting the line with the bug. Thus, the articles based on SZZ algorithm may be based on a wrong assumption. However, a larger number of tickets has to be studied to ascertain that our findings can be generalized. Our idea is contribute with this study in the knowledge about the responsibility practiced by the previous commit in a bug, helping to the software engineering community in the code review process. FIXME: aclarar esta frase que no me queda clara: Our idea is propose a reinterpretation in the responsibility practiced by the previous commit in a bug.

In the next months, we will carry out another empirical study with a larger sample size to prove that our results can be generalized. In this way, with a larger population we may be able to find certain patterns that we are now omitting. Furthermore, we extend this study to other projects to ascertain that what we have found occurs regardless of the project analyzed, because the OpenStack may be a special project, where the code is evolving continuously. In fact, we can access a huge database of other projects such as Eclipse or Mozilla that will allow us analyze this assumption, and where we will use statistical analysis to compare the results obtained in the different projects.

In addition, we have developed a tool that automatizes the first stage, displaying all the information used in the analysis in a web page[5], decreasing the time used in the analysis for the first stage. The future work with the tool will focus on improving and automating the second stage.

6. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

7. REFERENCES

[1] K. Herzig, S. Just, and A. Zeller. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 392–401. IEEE Press, 2013.

[2] E. W. Myers. Ano (nd) difference algorithm and its variations. *Algorithmica*, 1(1-4):251–266, 1986.

[3] E. Ukkonen. Algorithms for approximate string matching. *Information and control*, 64(1):100–118, 1985.

[5] http://bugtracking.libresoft.es