# 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 correspooding 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

- Codigo que era correcto deja de serlo - En el momento en el que se introdujo fue un error - Â£CÃşdigo erroneo o hubo otro cambio? - DÃ■ficil conocer el codigo en todo el contexto

We refer to such *previous commit* that the commit immediately before the bug fix.

In this paper, we attempt to address the following research question regarding who introduced the bug in the source code.

- RQ1: How I can know that a change was done to fix a bug in the source code? How I can identify them?
- RQ2: In which cases the previous commit/s are responsible to cause the bug?

The remainder of this paper is structured as follows. First, we present the motivations that support our study, explaining the current body of knowleadge in section 2. Then, Section 3 describes the methodology used to identify the moment in which the bug was inserted in the source code, followed by the results obtained after applaying our approach to OpenStack in Section 5. Section 6 discusses potential applications and improvements of our approach. Section 7 reports threats to validity. Finally, Section 8 concludes the article.

## 2. RELATED WORK

First SZZ
Second Improvement of SZZ
Then SZZ revisited
Then Buginnings
Finally our idea.

## 3. METHODOLOGY

We extract all the data necesary to analyze when the bug was inserted from the issue tracking system and the code review system used by the project we are analyzing. OpenStack uses Launchpad[1] and Gerrit[2].

The Launchapd of each project works with issues reports called tickets, which describe bug reports, feature requests, maintenance tickets, and even design discussions. But, in our study we are only interested in those tickets that describe a bug report and, in addition, have been closed with a merged in the code source to fix the bug. In this bug reports we can find a comment with the link of Gerrit where the bug was fixed. Is in Gerrit where we can see all the patchsets proposed and the comments done by the reviewers.

### 3.1 Fist Stage: The Filtering

In our approach, first of all we must identifying which of those issues extracted from Launchapd are bug reports. But, this is not a trivial task and we performed this identification using a web tool[3] development to provide the researcher with all the relevant information needed to decide if an issue corresponds to a bug report or not. The tool uses information




*MSR '16 Austin, Texas USA*



DOI: 10.475/123_4


[1] https://launchpad.net/openstack
[2] https://review.openstack.org/
[3] bugtracking.libresoft.es

| After Fix Bug | | Fix-inducing (Before fix Bug) | | Before fix-inducing | |
|---|---|---|---|---|---|
| 31f208423 711) | if rescue_auto_disk_config is None: | 31f208423 711) | if rescue_auto_disk_config is None: | 31f208423 701) | 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 702) | 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 703) | "rescue image_properties. Setting value to %s", |
| e30b45f69 714) | auto_disk_config, instance=instance) | e30b45f69 714) | auto_disk_config, instance=instance) | e30b45f69 704) | auto_disk_config, instance=instance) |
| 31f208423 715) | else: | 31f208423 715) | else: | | |
| 20847c25a 716) | auto_disk_config = strutils.bool_from_string( | 31f208423 716) | auto_disk_config = rescue_auto_disk_config | | |
| 20847c25a 717) | rescue_auto_disk_config) | | | | |
| (1) | | (2) | | (3) | |

**Figure 1: 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 2: Description of the bug-fix commit when the previous commit caused the bug**

```
Update default_project param on create user

In keystone v3, the parameter to create user for the the default project
has changed from project to default_project and is no longer honored and
throws an exception. Also passing in '' rather than None causes keystone
issues, so moving to None.

Closes-Bug: #1478143
Change-Id: I73423433a42bf46769065a269a3c35f27175f185
```

**Figure 3: Description of the bug-fix commit when the previous commit dind't cause the bug**

extracted automatically from the project repositories, and offers a web-based interface which allows for collaboration, traceability and transparency of the identification of bug reports.

During the identification of the issues we have to take into account the next parameters for each ticket, 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*.

As result of analyze the tickets, main differences extracting data and to classifying tickets were found. So, we agree to follow the next four criteria:

- When there are only test files in the ticket, we classified it as note being a bug report. Test files in a ticket will not be analyzed, they are indispensables and used as testing method to determine whether the code is fit for use. Sometimes the developers inserted the bug only in the test files, in these cases the ticket was not considered as bug report because the software works as expected only failed the test.
- When the title described optimization, deletion of a dead code or the implementation of new characteristics, our criteria indicated that it was not a bug report because there is no failure.
- When the title described the program as not working as expected, our criteria indicated that it was a bug report.
- When the title described that updates were required, our criteria indicated that it was a bug report. We consider all tickets that require updating as bug reports, because updating a software hints to the software not operating as expected.

Sometimes we were unable to answer all the questions due to having insufficient data or because of the complexity of the issue. In this case, the ticket was classified into the *Undecided* group.

## 3.2 Second Stage: Responsability of Previous Commit

The next part is focusing on analyzing the previous commit in the *Bug Report* group. For that, we had to analyze the lines involved in the bug fix, in the commit parent of the bug fix commit, and be sure that the lines was inserted/modified in 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. Also, we used *diff* to see all the differences between two files, in our case, the file is going to be the same but in different moment inside the control version system used.

Te procedure following in each file involved in a fix bug is show in the figure 4, and describes below;

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.

Finally we need to discard some noise presents in our final resulst according to the responsability of the previous commit inserting the bug in the code source. Due to they were not responsibles for cause the bug, we delete those previous commit which presents the following criteria;

- Blank lines
- Format changes
- Copied lines
- Changes in the comment.
- Updatings in the version of a file.

# 4. EVALUATION

We validate our methodology anlayzing 459 tickets in OpenStack. 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 [4]. Futhermore, all history is saved and available in a version control system, being able to access to its issue tracking system [5] and the source code review [6]

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 | - | - |
| Fuel | - | - |
| Netron | - | - |
| Openstack-manuals | - | - |
| Horizon | - | - |
| Cinder | - | - |
| Keystone | - | - |
| Heat | - | - |
| Glance | - | - |
| Tempest | - | - |

In this four projects we analized the relationship of bug fixes with their previous commits. In order to identify the moment in which the bug was injected into the source code.

[4]http://activity.openstack.org/dash/browser/
[5]https://launchpad.net/openstack
[6]https://review.openstack.org/

**Table 2: Classification statistics of each researcher**

| | Bug Report | Not Bug Report | Undecided | Total |
|---|---|---|---|---|
| R1 | (184) 55% | (115) 34% | (35) 11% | 334 |
| R2 | (188) 76% | (54) 22 % | (7) 3% | 249 |
| R3 | (188) 56% | (116) 35% | (30) 9% | 334 |
| Finally | (209) 72% | (74) 25% | (9) 3% | 292 |

The first stage we did automatically using the tool and a double bind between three researchers, three PhD student included me. The second stage was done manually and only me was involved analizing this relationship.

# 5. RESULTS

We extracted a total of 459 different tickets from the Launchpad of the four principal projects in OpenStack, 125 tickets from Nova, 125 tickets from cinder, 125 tickets from Horizon and 84 tickets from Neutron. These tickets first, were analized to identify those ones which were real bug reports. And secondly, each file was analyzed to obtain which previous commit inserted the bug causing the failure of the system and reporting the bug report.

## 5.1 Fist Stage

We classify a total of 459 tickets using the tool. 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 shows the percentage of each researcher after analyzing the tickets, and the number of tickets classifyed identically by two different researchers. Obtaining that the researchers R1 and R2 had a similar data in their results, whereas research R2 got results significantly differents with a higher number of tickets classifyed as Bug Report.

Finally, the researchers identifyed in the same way 292 tickets, that is, their results matched in a 70% of the cases. Obtaining 209 tickets classifyed as Bug report, 74 tickets classifyed as Not Bug Report and 9 tickets classifyed as Undecided.

Also we had measured the concordance in the classification of each developer according to the project analized. The table 3 shows that the concordante got between all the three research was very similar, around a 70%. Furthermore, the concordance form each researcher with the rest alwas was up to 60%.

**Table 3: Concordance between each developer in each repository**

| | Nova | Cinder | Horizon | Neutron | Total |
|---|---|---|---|---|---|
| R1 and R2 | (44) 70% | (40) 77% | (37) 60% | - | 68% |
| R1 and R3 | - | (46) 73% | (48) 76% | (26)62% | 71 % |
| R2 and R3 | (41) 66% | (10) 100% | - | - | 71% |

## 5.2 Second Stage

At this stage, we analyzed the 209 we got a list with all the previous commits and we were be able to classifyed the previous commit/s as Responsible, Not Responsible or Undecided, taken into account that the bug could be inserted

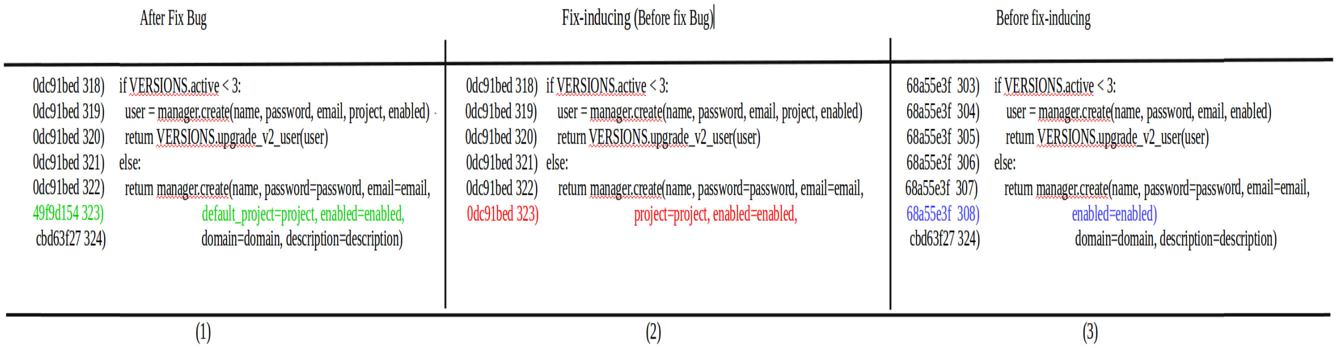


**Figure 4: Process to discover the commit that caused the bug**

in different lines of different commits, but not everyone had to be responsible for the commit, sometimes the previous commit copied lines from its previous commit or inserted comments and blank spaces. In fact, the responsible can be only one of them, more than one or maybe none.

We identifyed a total of 348 previous commits which can be responsible for inserting the line containing the bug. After analizing the Bug Reports and their previous commits and discarded 40 of them because were noise, Table 4, we got that 152 previous commit were responsibles to cause the bug whereas 114 previous commit had not any responsability in the failure of the system, and only in 42 previous commits we were unable to identify the cause of the bug.

**Table 4: Responsability of each previous commit before and after deleting the noise in the results**

| | Before Deleting Noise | After Deleting Noise |
|---|---|---|
| Responsible | 152 | 152 |
| Not responsible | 154 | 114 |
| Undecided | 42 | 42 |

Futhermore, focusing on how many previous commits presented each Bug Report, we obtained that 131 had one previous commit implicated, whereas 58 had more than one previous commit implicated in their file/s. According to Table 5, from the 131, we obtained that 65 of them inserted the bug, but 30 of them were not responsibles in the failure of the system. And from the 58 which had more than one previous commits, we obtained in total 189 previous commit, where 86 of them were respnsibles and 82 were not responsibles.

**Table 5: Probability of cause the bug depending on how many previous commits had the bug report**

| | One previous commit | More than one previous commit |
|---|---|---|
| Responsible | 65 | 86 |
| Not responsible | 30 | 82 |
| Undecided | 36 | 11 |

Also, we looked the distribution of the previous commit in each Bug Report, Table 6, we observed that the most commun distribuion in the relationship between previous commit and bug report is one previous commit per Bug Report, following by the second commun distribution, two previous commit per Bug Report.

Finally we wanted to know the responsability preactised by each previous commit in the failure of a system, in other words, we were interested in analize from those cases where exists more than one previous commit, how many of them inserted the bug in the code source, Table **??**. We obtained that in 8 Bug Reports all the previous commits were responsibles, in 30 Bug reports at least one of their previous commit caused the bug and in 11 bug report none of their previous commits inserted the bug.

## 6. DISCUSSION

- Como se responden las RQ1 y RQ2 - No todos los casos son tan claros como los mostrados en los ejemplos .... - Casuistica del common juidment - Hemos sido conservadores -Hemos utilizado la herramienta porque es un proceso complicado 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?

## 7. THREATS TO VALIDITY

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

Table 6: Distribution of number of previous commit per Bug Report in each project

| | One previous commit | two previous commit | three previous commit | four previous commit | +five previous commit |
|---|---|---|---|---|---|
| Neutron | 11 | 3 | 2 | 2 | 0 |
| Horizon | 39 | 8 | 3 | 2 | 4 |
| Nova | 44 | 5 | 2 | 4 | 4 |
| Cinder | 37 | 9 | 6 | 2 | 2 |
| Total | 131 | 25 | 13 | 10 | 10 |

Table 7: Probability of cause the bug depending on how many previous commits had the bug report

| | two previous commit | three previous commit | four previous commit | +five previous commit | Total |
|---|---|---|---|---|---|
| All Responsible | 4 | 3 | 0 | 1 | 8 |
| At least one responsible | 9 | 7 | 5 | 9 | 30 |
| None Responsible | 4 | 2 | 4 | 1 | 11 |
| Undecided | 1 | 0 | 1 | 0 | 2 |

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.

# 8. CONCLUSIONS

# 9. ACKNOWLEDGMENTS

We thank the two phd students, Dorealda Dalipaj and Nelson Sekitoleko, that participated differentiating Bug report from the others. Also, we thank Bitergia [7] to explain its available database of OpenStack. Finally,thanks the Spanish Goverment because all authors are funded in part by it, through project TIN2014-59400-R.

[7]http://bitergia.com/

# 10. REFERENCES

[1] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein. The missing links: bugs and bug-fix commits. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 97–106. ACM, 2010.

[2] M. Fejzer, M. Wojtyna, M. Burzańska, P. Wiśniewski, and K. Stencel. Supporting code review by automatic detection of potentially buggy changes. In *Beyond Databases, Architectures and Structures*, pages 473–482. Springer, 2015.

[3] 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.

[4] A. Hindle, D. M. German, and R. Holt. What do large commits tell us?: a taxonomical study of large commits. In *Proceedings of the 2008 international working conference on Mining software repositories*, pages 99–108. ACM, 2008.

[5] D. Izquierdo-Cortazar, A. Capiluppi, and J. M. Gonzalez-Barahona. Are developers fixing their own bugs?: Tracing bug-fixing and bug-seeding committers. *International Journal of Open Source Software and Processes (IJOSSP)*, 3(2):23–42, 2011.

[6] S. Kim, E. J. Whitehead Jr, and Y. Zhang. Classifying software changes: Clean or buggy? *Software Engineering, IEEE Transactions on*, 34(2):181–196, 2008.

[7] S. Kim, T. Zimmermann, K. Pan, and E. J. Whitehead Jr. Automatic identification of bug-introducing changes. In *Automated Software Engineering, 2006. ASE'06. 21st IEEE/ACM International Conference on*, pages 81–90. IEEE, 2006.

[8] S. Koch. *Free/open source software development.* Igi Global, 2005.

[9] D. MacKenzie, P. Eggert, and R. Stallman. *Comparing and Merging Files with GNU diff and*

*patch*. Network Theory Ltd., 2003.

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

[11] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen. Multi-layered approach for recovering links between bug reports and fixes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 63. ACM, 2012.

[12] K. Pan, S. Kim, and E. J. Whitehead Jr. Toward an understanding of bug fix patterns. *Empirical Software Engineering*, 14(3):286–315, 2009.

[13] V. S. Sinha, S. Sinha, and S. Rao. Buginnings: identifying the origins of a bug. In *Proceedings of the 3rd India software engineering conference*, pages 3–12. ACM, 2010.

[14] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.

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

[16] C. Williams and J. Spacco. Szz revisited: verifying when changes induce fixes. In *Proceedings of the 2008 workshop on Defects in large software systems*, pages 32–36. ACM, 2008.

[17] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 15–25. ACM, 2011.

[18] Z. Yin, D. Yuan, Y. Zhou, S. Pasupathy, and L. Bairavasundaram. How do fixes become bugs? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 26–36. ACM, 2011.

[19] T. Zimmermann, S. Kim, A. Zeller, and E. J. Whitehead Jr. Mining version archives for co-changed lines. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 72–75. ACM, 2006.

[20] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. *Software Engineering, IEEE Transactions on*, 31(6):429–445, 2005.