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

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

The remainder of this paper is structured as follows. First, we present the motivations that support our study, explaining the current problem in section 2. Then, Section 3 describes our empirical study in an open source project such as OpenStack, followed by the results in Section 4. Section 5 discusses potential applications and improvements of our approach. Section 6 reports threats to validity. Finally, Section 7 presenst the actual body of knowdeadge about who caused the bug and Section 8 concludes the article.

MSR '16 Austin, Texas USA

DOI: 10.1145/1235

2. THE CURRENT ASSUMPTION

- Problem of missclassification
- Problem of little empirical evidence about the hypotesis.

3. 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 [3] and this should cause bias in our final results.

3.1 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]

Each ticket has an id unique in Launchpad, which will be referred as *n_ticket*. Given its id, all information about the bug is publicly available in the issues tracking system[5]. OpenStack uses Gerrit as code review system to see the evolution of each bug report[6]. Specifically we found the code review for each *'n_ticket'*[7], where *'n_review'* is the review unique id for each *'n_ticket'*. In the code review all the information about the patchset that fixed the bug is shown.

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*.

The researchers found main differences to extract data or to classify. In case of disagreement about the ticket classification among researchers, four criteria were used:

- 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 a new feature, our criteria indicated that it was not a bug report. New features are not considered bug reports, because there is no failure. The optimization, deletion of a dead code or the implementation of new characteristics are included in this criteria.
- When the title described the program as not working as expected, our criteria indicated that it was a bug report. Sometimes the bug is not in a main function which prevents `Cinder` from working, because a developer has considered it an important bug, although not relatively important. But it is not working as he/she expected.
- 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.

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

The table 2 shows that the concordance of the developers is high but, also demonstrate the difficulty to classify tickets as bug report or as not bug report, because each developer can have different ideas about a specific ticket. The concordance between the developers could be higer if they were expert in the project.

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.

3.1.1 BugTracking: The tool used

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 is available at[8] and 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 webpage provide us different functionalities depend on in which tab we are, next we explain these functionalities.

1. Tab Repository: In this tab we choose which repository of OpenStack we want analyze. Currently the tool supports the four principal repositories: Cinder, Nova, Neutron and Horizon, in which there is more activity.
2. Tab Analyze: Is the Tab showed in 1 where the user select/insert an identifyer of a ticket and analyze with all the data displayed if the ticket are a bug report or not.
3. Tab Statistics: This tab collects the data saved in the user's repository in GitHub after analyze each ticket, and displays a table with the number of tickets classified as *Bug Report,Not Bug Report* and *Undecided* in

[4]https://bugs.launchpad.net/NameOfRepository
[5]https://bugs.launchpad.net/cinder/+bug/n_ticket
[6]https://review.openstack.org/
[7]https://review.openstack.org/#/c/n_review
[8]bugtracking.libresoft.es

Table 2: Concordance between each developer in each repository

	Nova	Cinder	Horizon	Neutron	Total
D1 and D2	(44/63) 70%	(40/52) 77%	(37/62) 60%	-	68%
D1 and D3	-	(46/63) 73%	(48/63) 76%	(26/42) 62%	71 %
D2 and D3	(41/62) 66%	(10/10) 100%	-	-	71%

the repository and from each developer involved in the analysis, which name has to be selected previously.

4. Tab Modify: In this tab, the user can see all his data saved in the GitHub repository and modify the content of the file that he wants, in case of have inserted a mistake during the analysis.

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

3.2 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.

At this stage, we got a list with all the previous commits, which could be where the bug was introduced. We identifyed a total of 348 previous commits which can be 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 to be responsible for the commit, sometimes the previous commit copied lines from its previous commit or inserted comments and blank spaces. After the analysis the responsible can be only one of them, more than one or none.

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, according to the responsability of the previous commit in the bug fix:

1. It is responsible, when we are sure that the commit inserted the line with the bug.

2. It is not responsible, when we are sure that the commit dind't insert the line with the bug.

3. Undecided, when we cannot be sure if the commit is responsible due to our few knowleadge about the project or when the bug fix only added lines, complicating the analysis.

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.

After analayzed all the previous commit involved in the 209 Bug reports, we have discovered that not all the previous commits inserted the bug. This expecific project, which is continuosly evolving, presents some cases in which the bug was introduced by other actions. A clear example is the changes in called APIs, after the addition of a new feauture the called API fails, due to it requires an extra argument, but in the API there is not any bug.

Other example is shows in the commit log of a bug fix 5, the name of a variable changed in the new version causing a failure. This change due to the new requirements in the version doesn't implicate that the previous commit was who inserted the bug, because until the new version the system doesn't fail.

3.2.1 Discarting False Positives

The classification of the previous commits according to its responsability has some noise that we had to delete. These previous commits that changed the format, added blank lines, changed comments or copied lines from the previous commit were discarted due to they were not responsibles for cause the bug.

4. RESULTS

4.1 Fist Stage

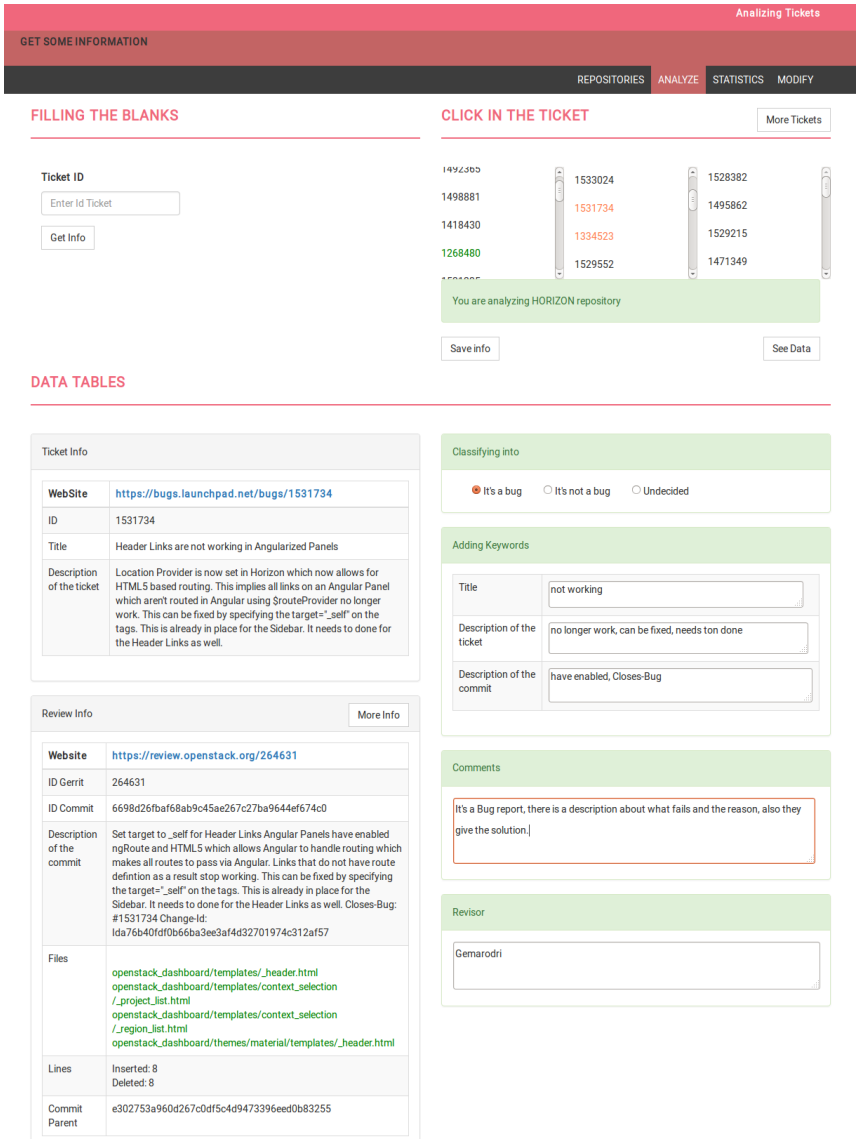

Figure 1: Screenshot of Analyze Tab

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

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 3 show the percentage of each researcher after analyzing the tickets, 417 tickets were analyzed by two different researchers,

4.2 Second Stage

58 tickets with more than one previous commit and 131 with only one previous commit

5. 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?

6. THREATS TO VALIDITY

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

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.

7. RELATED WORK

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

8. CONCLUSIONS

9. ACKNOWLEDGMENTS

We thank the two phd students, Dorealda Dalipaj and Nelson Sekitoleko, that participated in the tickets differentiating Bug report from the others. Also, we thank Bitergia [9] to explain its available database of OpenStack.

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.

[9] http://bitergia.com/

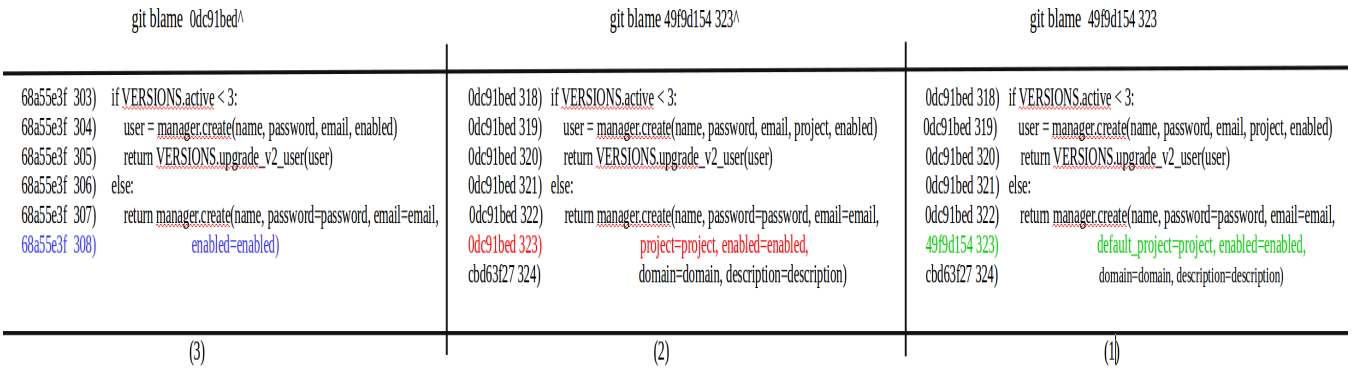

Figure 4: Process to discover the commit that caused the bug

Table 3: 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

```
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 5: Description of the bug-fix commit

Table 4: 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

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%

[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.