The Symptoms, Causes, and Repairs of Workarounds in Apache Issue Trackers

Aoyang Yan, Hao Zhong, Daohan Song, and Li Jia
Shanghai Jiao Tong University, China
{xiaoyan9894,zhonghao}@sjtu.edu.com,251287012@qq.com,insanelung@sjtu.edu.cn

ABSTRACT

In issue tracker systems, a bug report can be resolved as *workaround*. Since the definition of workarounds is vague, many research questions on workarounds are still open. For example, what are the symptoms of bugs which are resolved as workarounds? Why do a bug report have to be resolved as workarounds? What are the repairs and impacts of workarounds? In this paper, we conduct an empirical study to explore the above research questions. In particular, we analyzed 221 real workarounds that were collected from 88 Apache projects. Our results lead to ten findings and our answers to all the above questions. Our findings are useful to understand workarounds and to improve software projects and issue trackers.

1 INTRODUCTION

Issue tracker systems (*e.g.*, JIRA [1]) are widely used to manage bug reports. A bug report typically is resolved as *fixed*, *invalid*, *duplicated* or *won't fix*. The definitions of the above resolutions are clear. For example, a *fixed* bug report denotes that the problem is resolved. Users can make their choices based on these resolutions. For example, if the problem of a bug report is serious and the bug is *fixed*, users can update to a newer version to avoid the problem.

Although it is less known, a bug report can be resolved as a *workaround*. By its literal definition, a workaround is a bypass to solve or avoid a problem, when the most obvious solution is not possible. After reading this definition, people may still raise questions besides what are already reported in bug reports. For example, are the problems actually fixed or not? If a problem is literally avoided, does it become a technical debt [14]? What is the trouble of directly resolving the problem? Indeed, even the very definition of workarounds is unclear. When the prior studies (*e.g.*, [9]) introduce the work flow of issue trackers, they did not mention workarounds at all. The questions hinder users from making a good decision, even if the user knows that a bug is fixed as workarounds.

To deepen the knowledge on workarounds, Song *et al.* [11] conduct the first empirical study on workarounds. Compared with their study, we further refine the categories of workarounds, define workarounds, and present the associations of the categories.

2 METHODOLOGY

Dataset. In total, we collected 221 workarounds, and to ensure the diversity of the dataset, these workarounds were collected from 88 Apache projects. The size of our dataset is comparable to those of other related empirical studies. For example, Zhang *et al.* [15] analyzed 175 tensorflow [2] bugs.

Protocol. In our study, we manually inspected all the workarounds. Our analysis protocol is as follows: First, for each bug report, We read the website of the project to understand its overall functionality and users (*e.g.*, programmers or end users). This analysis is useful to understand the scope of bug reports. Second, we read the title, description and discussions of a bug report to understand its symptoms. Third, we read the discussions of a bug report to understand its causes and repairs. A programmer marks a bug report as a workaround, and the comments from this program are typically useful to understand why a bug is resolved as a workaround. If bug reports have related issue reports and pull requests, we further read their related issue reports and pull requests. For each bug report, we checked its code from its github repository and searched for commits that resolve a bug with its issue number. If such commits are found, we read their code changes to determine their types of causes and repairs. The categories of symptoms are predefined by prior studies [10, 12, 13, 15], but the categories of causes and repairs are defined by ourselves, since no prior study has built such taxonomy for workarounds. Following this protocol, we inspected the bugs independently, and compared the results for differences. In this study, we tried to resolve all inconsistencies. If we could not come to an agreement, we contacted their programmers by sending emails or directly discussing on its bug report.

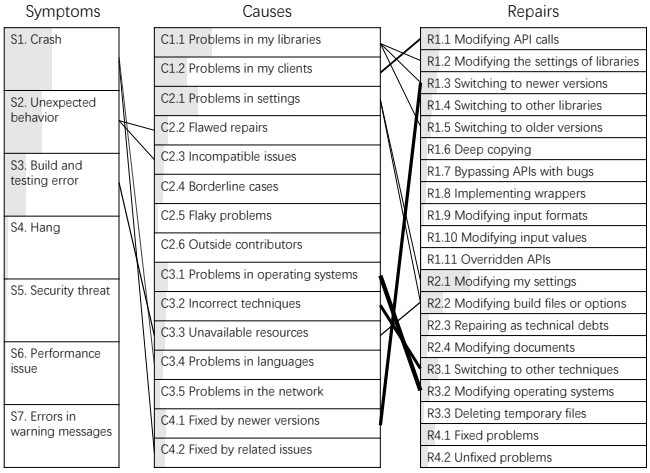

Figure 1: The overall results

3 EMPIRICAL RESULT

Our study answers the following research questions:

RQ1. Which symptoms are resolved as workarounds?

Most workarounds fix crashes (41.63%) and unexpected behaviors (32.58%). For example, a workaround [3] describes a crash: "DNS Packets with dns.flags.rcode=1 cause ml_ops.sh to crash". In the standard process, programmers determine that a crash [4] is caused by a wrong way to retrieve values from a table, and fix the code to retrieve the correct values.

RQ2. Why are workarounds introduced?

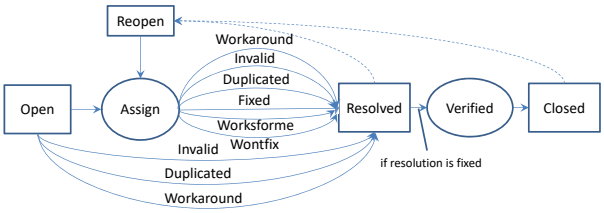

Figure 2: The life cycle of a bug report with workarounds

In total, 34.84% workarounds fixed problems on their own projects, but their repairs are imperfect (*e.g.*, flawed repairs 5.88%), difficult to reproduce (*e.g.*, flaky problems 1.36%), modifications on non-source code (e.g., settings 19.46%), or solved by outside contributors. The problems of about half workarounds are across projects (38.01%) or reside in environments (21.27%). In a few cases, a bug report is marked as workaround, because its problem is already fixed (9.50%).

RQ3. How do workarounds repair bugs?

In most cases, libraries cannot be modified to repair problems. As workarounds, programmers can switch whole libraries (*e.g.*, with a newer version 7.24%), switch APIs with problems (*e.g.*, overridden 0.90%), or switch the way to call APIs (8.60%). When repairs happen in a project where a bug is reported, they often modify settings (21.72%), build files (9.95%), and documents (2.71%). Meanwhile, the repairs on source files are technical debts (3.62%). As workarounds, programmers can recommend switching to other techniques (7.69%), modifying their own operation systems (4.07%), or deleting temporary files (1.36%). A workaround can have no repairs due to two different reasons: (1) some bugs are already fixed (6.33%) and (2) some cannot be repaired at present (3.17%). For example, a bug report [5] of `Zeppelin` [6] complains a hang. To resolve the problem, a programmer submits a pull request [7], and proposes to change the default input of a method. However, the pull request is not approved, because the problem already disappears in a newer version. As a result, the pull request is deleted, and no repairs are applied.

RQ4. What are the associations among the symptoms, causes and repairs of workarounds?

Figure 1 shows the associations of symptoms, causes, and repairs. In this Figure, each column denotes one of the above three dimensions, and a node of a dimension denotes a category of the dimension. An edge between two nodes denotes a association between the two corresponding categories, and a thicker edge denotes a stronger association. We find that most associations between causes and repairs are straightforward. For example, if the operating system causes a bug, a typical workaround is to select alternative operating systems. Crashes are fixed as workarounds, often because they are fixed problems or their problems reside in environments. Unexpected behaviors are fixed as workarounds, often because their repairs have flaws.

RQ5. How to define workarounds?

Figure 2 presents the work flow with workarounds. The problems of 90.50% workarounds are not fixed when they are reported, and their definition is as follows:

Definition 1. *A workaround is a type of bug reports, whose problems are bypassed, because their problems are infeasible or difficult to be handled in the standard process.*

The problems of 9.50% workarounds are already fixed when they are reported, and the definition of these workarounds is as follows:

Definition 2. *A workaround is a type of bug reports, whose problems are already fixed in new versions or related issues.*

Herzig *et al.* [8] find that researchers and programmers have different definitions on the types of issue reports. We find that even programmers themselves have different definitions on workarounds. The inconsistencies can be confusing, and some workarounds can be better marked as a different type of transitions. The problem can be resolved, if issue trackers define finer status. For example, if the problem of a bug report is fixed in new versions or related issues, it can be better marked as *fixed in other locations* than workarounds. More details are presented on our website:

https://github.com/tetradecane/Workaround_poster_website

4 CONCLUSION AND FUTURE WORK

Our future work plan is as follows:

1. Presenting more examples and details. Due to space limit, we present limited examples and brief analysis protocol in this paper. When extending our poster to a full paper, we will present more illustrative examples, and write detailed protocols.

2. Cleansing workarounds. We find that programmers can mark two different types of issue reports as workarounds. In our rejected submissions, some researchers criticize that the second definition is inconsistent with what researchers anticipate, and they suggest that we shall remove them to avoid the pollution from such so-called workarounds. In future work, we plan to follow their advices, and present the results after cleansing workarounds.

3. Interpreting our findings. We will present actionable advises based on our findings.

ACKNOWLEDGEMENT

We appreciate the reviewers for their comments. This work is sponsored by a CCF-Huawei Innovation Research Plan (No. CCF202 l-admin-270-202111). Hao Zhong is the corresponding author.

REFERENCES

[1] 2020. https://issues.apache.org/jira. (2020).
[2] 2020. https://github.com/tensorflow/. (2020).
[3] 2020. https://issues.apache.org/jira/browse/SPOT-26. (2020).
[4] 2020. https://issues.apache.org/jira/browse/SPOT-238. (2020).
[5] 2020. https://issues.apache.org/jira/browse/ZEPPELIN-3175. (2020).
[6] 2020. http://zeppelin.apache.org/. (2020).
[7] 2020. https://github.com/apache/zeppelin/pull/2733. (2020).
[8] Kim Herzig, Sascha Just, and Andreas Zeller. 2013. It's not a bug, it's a feature: how misclassification impacts bug prediction. In *Proc. ICSE*. 392–401.
[9] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. 2009. Improving bug triage with bug tossing graphs. In *Proc. ESEC/FSE*. 111–120.
[10] Li Jia, Hao Zhong, Xiaoyin Wang, Linpeng Huang, and Xuansheng Lu. 2020. An Empirical Study on Bugs inside TensorFlows. In *Proc. DASFAA*. to appear.
[11] Daohan Song, Hao Zhong, and Li Jia. 2020. The Symptom, Cause and Repair of Workaround. In *Proc. ASE*. 1264–1266.
[12] Lin Tan, Chen Liu, Zhenmin Li, Xuanhui Wang, Yuanyuan Zhou, and ChengXiang Zhai. 2014. Bug characteristics in open source software. *Empirical Software Engineering* 19, 6 (2014), 1665–1705.
[13] Ferdian Thung, Shaowei Wang, David Lo, and Lingxiao Jiang. 2012. An Empirical Study of Bugs in Machine Learning Systems. In *Proc. ISSRE*. 271–280.
[14] Meng Yan, Xin Xia, Emad Shihab, David Lo, Jianwei Yin, and Xiaohu Yang. 2018. Automating change-level self-admitted technical debt determination. *IEEE Transactions on Software Engineering* 45, 12 (2018), 1211–1229.
[15] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An empirical study on TensorFlow program bugs. In *Proc. ISSTA*. 129–140.