The Symptom, Cause and Repair of Workaround

Daohan Song, Hao Zhong, and Li Jia
Department of Computer Science and Engineering, Shanghai Jiao Tong University
251287012@qq.com,zhonghao@sjtu.edu.cn,insanelung@sjtu.edu.cn

ABSTRACT

In software development, issue tracker systems are widely used to manage bug reports. In such a system, a bug report can be filed, diagnosed, assigned, and fixed. In the standard process, a bug can be resolved as *fixed*, *invalid*, *duplicated* or *won't fix*. Although the above resolutions are well-defined and easy to understand, a bug report can end with a less known resolution, *i.e.*, *workaround*. Compared with other resolutions, the definition of workarounds is more ambiguous. Besides the problem that is reported in a bug report, the resolution of a workaround raises more questions. Some questions are important for users, especially those programmers who build their projects upon others (*e.g.*, libraries). Although some early studies have been conducted to analyze API workarounds, many research questions on workarounds are still open. For example, which bugs are resolved as workarounds? Why is a bug report resolved as workarounds? What are the repairs of workarounds? In this experience paper, we conduct the first empirical study to explore the above research questions. In particular, we analyzed 221 real workarounds that were collected from Apache projects. Our results lead to some interesting and useful answers to all the above questions. For example, we find that most bug reports are resolved as workarounds, because their problems reside in libraries (24.43%), settings (18.55%), and clients (10.41%). Among them, many bugs are difficult to be fixed fully and perfectly. As a late breaking result, we can only briefly introduce our study, but we present a detailed plan to extend it to a full paper.

ACM Reference Format:
Daohan Song, Hao Zhong, and Li Jia. 2020. The Symptom, Cause and Repair of Workaround. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3324884.3418910

1 INTRODUCTION

In software development, issue tracker systems (*e.g.*, JIRA [1]) are widely used to manage bug reports. As introduced by Zeller [14] and Jeong *et al.* [11], from tracker systems, a bug report can be filed, diagnosed, assigned, and fixed. In the standard process, a bug report typically is resolved as *fixed*, *invalid*, *duplicated* or *won't fix*. The above resolutions are clearly defined, and users can make their choices based these resolutions. Although it is less known, a bug report can end as a *workaround*. By its literal definition, a workaround is a way in which a problem is resolved or avoided, when the most obvious solution is not possible. After reading this definition, the resolutions of workarounds still raise questions besides what are already reported in bug reports. For example, are the problems fixed or not? If a problem is literally avoided, is it a technical debt [13]? Why are bug reports resolved as workarounds at the first place? The questions hinder users from making a good decision, when handling workarounds.

As API libraries are widely used, most software projects are built upon libraries. If a workaround occurs in a library, it can lead to far-reaching impacts on its downstream projects, and a workaround on an API can reveal problems of its upstream projects. As API workarounds have notable impacts on programmers, researchers have conducted some early studies on API workarounds. For example, Bogart *et al.* [9] report that users can intentionally modify or bypass a problematic API as a workaround. As another example, Lamothe and Shang [12] summarize four patterns on API workarounds, by recreating the workarounds of 40 Stack Overflow posts. However, these studies did not touch workarounds as a research problem in the general software development. Even for API workarounds, many questions are still open after their studies. For example, besides the four patterns that were extracted by Lamothe and Shang [12], are there other patterns? To deepen the knowledge on workarounds, in this paper, we conduct the first empirical study on workarounds in general software devolvement. Our study explores the following research questions:

RQ1. What symptoms can be repaired as workarounds? This research question concerns which types of bugs can be resolved as workarounds, and we classify bug types by their symptoms.

RQ2. Why are bug reports resolved as workarounds? This research question concerns the causes of workarounds.

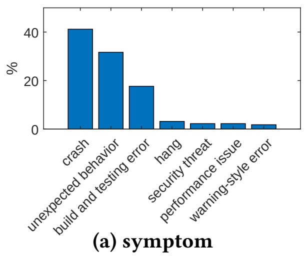
RQ3. How to repair bugs in the way of workarounds? This research question explores where the repairs are and why the repairs resolve a bug.

2 METHODOLOGY

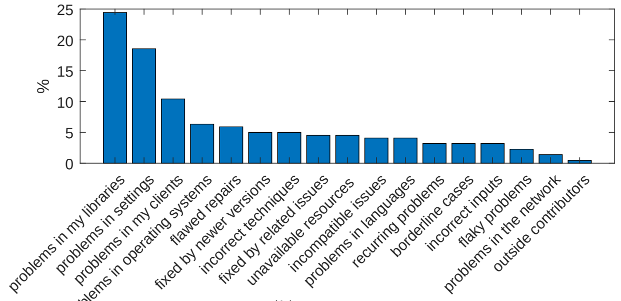
Dataset. To collect the dataset, we search the Apache JIRA [1] for bug reports whose resolutions are workarounds and statuses are either *resolved* or *closed*. In total, we collected 221 workarounds from 88 Apache projects. The size of our dataset is more than those of other related empirical studies. For example, Zhang *et al.* [15] analyzed 175 tensorflow [2] bugs, and Lamothe and Shang [12] analyzed 40 Stack Overflow posts.

Protocol. In our study, three authors manually inspected all the workarounds. All of them have rich experience in developing software and are familiar with bug tracking systems. Following the protocols, they inspected the bugs independently, and compared the results for differences. If any result is inconsistent, they contact

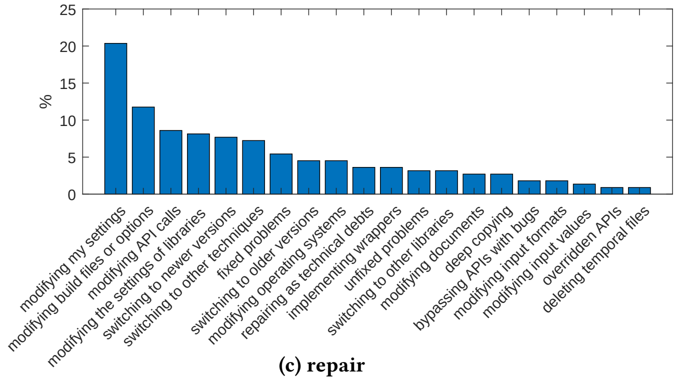
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components 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.
ASE '20, September 21–25, 2020, Virtual Event, Australia
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3418910



(a) symptom



(b) cause



(c) repair

Figure 1: The categories of workarounds

programmers by sending emails or directly discussing on bug reports. For example, they asked programmers on a workaround [3]. A programmer named Makoto Yui confirmed that this bug is caused by the memory requirements of `Spark` [4] and they have fixed this problem. In each RQ, we analyzed the 221 workarounds according to its corresponding protocol. In RQ1, RQ2, and RQ3, after we determine the categories of all workarounds, the percent of a category A is calculated as $\frac{N_A}{N_{All}}$, where N_A is the number of workarounds in category A, and N_{All} is the number of all workarounds, *i.e.*, 221. In RQ1, a workaround is put into exactly one category, so the sum of its percents is exactly one. In RQ2 and RQ3, several workarounds are classified into multiple categories, so the sums of their percents are more than one.

3 EMPIRICAL RESULTS

The details are listed on our anonymous project website:

https://anonymous.4open.science/r/7f52d2b5-4015-46b2-90e3-6750a078bccd/

Answer to RQ1: As shown in Figure 1a, crashes (41.18%), unexpected behaviors (31.67%), and build and testing errors (17.65%) are often resolved as workarounds.

Answer to RQ2: As shown in Figure 1b, most bug reports are resolved as workarounds, because their problems reside in libraries (24.43%), settings (18.55%), and clients (10.41%). We notice that these bug reports are often difficult to be fixed fully and perfectly.

Answer to RQ3: As shown in Figure 1c, the repairs of workarounds are more diverse than their symptoms and causes. Most workarounds modify the settings (20.36%), build files (11.76%), API calls (8.60%), and settings of libraries (8.14%).

As the first study on general workarounds, we have presented the distributions of workarounds for the first time. Even for API workarounds, we found some new results. Lamothe and Shang [12] reported four types of API workarounds by their repairs. Their first type, functionality extensions, corresponds to our "overridden APIs"; their second type, deep copies, corresponds to our "deep copying"; and we do not find the correspondences for their other two types such as multi-versions and unnecessary workarounds. From the perspective of causes, API workarounds are caused by problems in libraries. The patterns of the prior study [12] are less common in real projects (0.90% for functionality extensions and 2.71% for deep copies), because they can introduce incompatible issues. For example, if programmers modify the code [9] or deep copy the code [12] of a library, their client code becomes incompatible with new versions of the library, which can cause many problems. Instead, programmers more tend to switch to other versions of libraries or change their settings to resolve such bugs. For example, `NiFi Registry` [5] is a project to store and manage shared resources. A bug report [6] of `NiFi Registry` complains a crash with `JRE`. A programmer of `NiFi Registry` explains that `NiFi Registry` must call `JDK` rather than `JRE`, and resolves this problem. As another example, `Solr` [7] is an open source search platform. A bug report [8] of `Solr` complains a hang. A Programmer determines that a problem occurs when `JDBC` is stuck in the middle of a read, and finds that increasing `oracle.jdbc.ReadTimeout` can relieve the problem.

4 CONCLUSION AND RESEARCH PLAN

To extend our work to a full paper, our research plan is as follows:

1. Presenting illustrative examples. Due to space limit, we cannot present examples in this paper. When extending it to a full paper, we will present at least one illustrative example for each category, in that such examples are useful to understand workarounds.

2. Presenting more analysis details. We will write a separate protocol for each research question. The added details are useful to replicate our study on other datasets.

3. Analyzing workarounds in more details. We plan to enrich our study from three perspectives: (1) the correlations of symptoms, causes, and repairs [10]; (2) the evolution of workarounds; and (3) the relation between technical debts and workarounds.

4. Interpreting our findings. We will present actionable advises: (1) based on our findings, half of workarounds can be avoided by better interfaces, and (2) our results can provide insights on how to improve the infrastructures of OSS (*e.g.*, issue trackers).

ACKNOWLEDGEMENT

We appreciate the anonymous reviewers for their insightful comments. Hao Zhong is the corresponding author. This work is sponsored by the National Key R&D Program of China No. 2018YFC083050.

REFERENCES

[1] https://issues.apache.org/jira, 2020.
[2] https://github.com/tensorflow/, 2020.
[3] https://issues.apache.org/jira/browse/HIVEMALL-30, 2020.
[4] https://spark.apache.org/, 2020.
[5] http://nifi.apache.org/registry.html, 2020.
[6] https://issues.apache.org/jira/browse/NIFIREG-142, 2020.
[7] https://lucene.apache.org/solr/, 2020.
[8] https://issues.apache.org/jira/browse/SOLR-6209, 2020.
[9] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung. How to break an API: cost negotiation and community values in three software ecosystems. In *Proc. ESEC/FSE*, pages 109–120, 2016.
[10] J. Han, M. Kamber, and J. Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2011.
[11] G. Jeong, S. Kim, and T. Zimmermann. Improving bug triage with bug tossing graphs. In *Proc. ESEC/FSE*, page 111–120, 2009.
[12] M. Lamothe and W. Shang. When apis are intentionally bypassed: An exploratory study of api workarounds. In *Proc. ICSE*, page to appear, 2020.
[13] M. Yan, X. Xia, E. Shihab, D. Lo, J. Yin, and X. Yang. Automating change-level self-admitted technical debt determination. *IEEE Transactions on Software Engineering*, 45(12):1211–1229, 2018.
[14] A. Zeller. *Why Programs Fail: A Guide to Systematic Debugging*. Elsevier, 2006.
[15] Y. Zhang, Y. Chen, S. Cheung, Y. Xiong, and L. Zhang. An empirical study on TensorFlow program bugs. In *Proc. ISSTA*, pages 129–140, 2018.