What Are the Perception Gaps Between FLOSS Developers and SE Researchers?
A Case of Bug Finding Research

Yutaro Kashiwa(✉), Akinori Ihara, and Masao Ohira

Wakayama University, Wakayama, Japan
kashiwa.yutaro@g.wakayama-u.jp,{ihara,masao}@sys.wakayama-u.ac.jp

Abstract. In recent years, many researchers in the SE community have been devoting considerable efforts to provide FLOSS developers with a means to quickly find and fix various kinds of bugs in FLOSS products such as security and performance bugs. However, it is not exactly sure how FLOSS developers think about bugs to be removed preferentially. Without a full understanding of FLOSS developers' perceptions of bug finding and fixing, researchers' efforts might remain far away from FLOSS developers' needs. In this study, we interview 322 notable GitHub developers about high impact bugs to understand FLOSS developers' needs for bug finding and fixing, and we manually inspect and classify developers' answers (bugs) by symptoms and root causes of bugs. As a result, we show that security and breakage bugs are highly crucial for FLOSS developers. We also identify what kinds of high impact bugs should be studied newly by the SE community to help FLOSS developers.

AQ1

Keywords: Open source software · High impact bug · Interview

1 Introduction

The importance of FLOSS is increasing day by day, not only for personal use but also for enterprises to incorporate it into parts of their products. With the increase in the number of FLOSS users, use cases have also been expanding. As a result, lots of bugs are being reported to FLOSS projects. In the field of software engineering, many methods have been proposed to help FLOSS developers predict faults in modules, localize and repair faults in source code, and so on. However, these methods are too diverse for FLOSS developers to follow and effectively adapt to their projects as needed. Although several studies [28] provided a systematic review to mitigate the difficulty in understanding existing methods, they are neither beyond grouping existing studies nor necessarily in line with FLOSS developers' needs for support. Ideally speaking, we should try to thoroughly understand which bugs cause the most trouble for FLOSS developers and propose solutions to fix the bugs effectively and efficiently. In this

F. Bordeleau et al. (Eds.): OSS 2019, IFIP AICT 556, pp. 1–14, 2019.
https://doi.org/10.1007/978-3-030-20883-7_5

study, we try to reveal the gap between the way that FLOSS developers and researchers perceive bug finding and fixing, through interviews with 322 notable GitHub developers. In the interviews, we do not directly ask the developers about what troubleshooting tools are highly in-demand, but instead we ask them about what kinds of bugs causes them severe troubles, in order to enable us to precisely understand the problems they face. After the interviews, we discuss what kinds of high impact bugs are able to be solved or not by existing studies, in order to provide researchers with insights to come up with new solutions. This work is separate from general bug categorizations, and to the best of our knowledge, our work is the first bug categorization focusing on only high impact bugs. In this paper, we address the following research questions;

RQ1: What kinds of high impact bugs are mainly considered high impact by FLOSS developers?
RQ2: What kinds of high impact bugs do FLOSS developers encounter most frequently?
RQ3: What kinds of high impact bugs should be studied newly by the SE community in order to support FLOSS developers?

Our contributions to the study are as follows;

– We revealed symptoms and causes of bugs considered high impact by FLOSS developers, through interviews and manual inspections of bug reports.
– For FLOSS developers, we identified which kinds of high impact bugs are supported by previous studies.
– For researchers, we showed the area where FLOSS developers' needs are still not fulfilled.

2 High Impact Bugs

Over the past two decades, the SE community have dedicated considerable efforts to help software developers to predict faults in modules, localize and repair faults in source code, and so on. Although the existing, traditional studies had not thoroughly considered the characteristics nor the impacts of bugs, in recent years they began to tackle with the impacts of bugs on users and the development process. In what follows, we introduce some existing studies on finding and fixing high impact bugs.

A Security bug [8] can cause serious problems which often impacts on uses of software products directly. Since Internet devices (e.g., smartphones) usage is increasing every year, security issues of software products are of interest to many people. In general, security bugs are fixed as soon as possible.

A Performance bug [22] is defined as "programming errors that causes significant performance degradation." The performance degradation contains poor user experience, laggy application responsiveness, lower system throughput, and waste computational resources [18]. [22] showed that a performance bug needs more time to be fixed compared with a non-performance bug.

A Breakage bug [30] is a type of functional bug which is introduced into a product when the source code is modified to add new features or to fix existing bugs. Though it is well-known as regression, a breakage bug mainly focuses on regression in functionalities. A breakage bug causes problems which make available functions in one version unusable after releasing newer versions.

A Blocking bug [7] is a bug that prevents other bugs from being fixed. It is often caused due to a dependency relationship among software components. Since a blocking bug inhibit developers from fixing other dependent bugs, it can highly impact on developers' task scheduling since a blocking bug takes more time to be fixed [7] (i.e. a developer would need more time to fix a blocking bug and other developers need to wait for being fixed to resolve the dependent bugs).

A Surprise bug [30] is a new concept of software bugs. It can disturb the workflow and/or task scheduling of developers since it appears at unexpected times (e.g., bugs detected in post-release) and locations (e.g., bugs found in files are rarely changed in pre-release). As a result of a case study of a proprietary, telephony system which has been developed for 30 years, [30] showed that the co-changed files and the amount of time between the latest pre-release date and the release date can be good indicators of predicting surprise bugs.

A Dormant bug [4] is also a new concept on software bugs and is defined as "a bug that was introduced in one version (e.g., Version 1.1) of a system, yet it is not reported until after the next immediate version (i.e., a bug is reported against Version 1.2 or later)." [4] showed that 33% of the reported bugs in Apache Software Foundation projects were dormant bugs and were fixed faster than non-dormant bugs.

3 Study Design

3.1 Overview

In this study, we e-mail and ask notable developers in GitHub [9] to answer a questionnaire about high impact bugs. After aggregating collected responses, we show the developers' demographic information (**Q1**), and the distribution of the bugs that are considered high impact (**Q2-1**). As the questionnaire includes an open question (**Q2-2**) to tell us actual bug reports which caused troubles in the past, we manually inspect the bug reports and categorize them by symptoms.

3.2 Participant Selection

In order to select notable developers to invite to our interview in this study, we use *contribution* which represents the amount of the developer's commit activity to GitHub repositories and can be calculated with GitHub API [10]. First, we make a list of all repositories in GitHub and calculate the total number of contributions for each repository. Note that we only calculate contributions for the most committed repositories if the repositories have the same name, since forked repositories partly (sometimes largely) include the same commits from original repositories and we need to avoid multiple counts for the same contributions by the same developer. Next, the total contributions of each developer is calculated based on the selected repositories above, and we choose candidates who mark over 100 contributions. Finally, we sent e-mails to 22,228 candidate developers to ask them to participate in our interview.

3.3 Questionnaire

We prepared Google Forms for our interview which consisted of three questions to know the developers demography (**Q1**), one closed question to reveal a distribution of high impact bugs considered important by developers (**Q2-1**), and one open question to collect and further analyze actual reports on high impact bugs (**Q2-2**). The questionnaire has six more questions, but in this paper, we only focus on the five questions above due to the space limitations.

[**Q1. Profile**]

Q1-1 Your main project
Q1-2 Your experience with FLOSS development
Q1-3 Your motivation to participate in FLOSS development

[**Q2. High impact bugs**]

Q2-1 What kind of bug would be much more important to be fixed?
- a bug threatening systems' security (Security bug)
- a bug deteriorating system's performance (Performance bug)
- a bug blocking other bug fixes (Blocking Bug)
- a bug found in unexpected timing and location (Surprise bug)
- a bug introduced in older releases and found in a newer releases (Dormant bug)
- a bug introduced in a newer release and breaking functions of older releases (Breakage bug)
- others [free text]

Q2-2 Please tell us high impact bug(s) you encountered in the past.

3.4 Categorization of Bug Symptoms

Based on the responses of **Q2-2**, we collect actual bug reports from developers' projects and confirm the symptoms of the bugs, in order to discuss what techniques have been already proposed or that will be required to find and fix those high impact bugs. The first and second authors independently and manually inspect symptoms of actual high impact bugs and classify them by the card sort technique [23]. After the independent classification, the two authors discuss each classification result together and merge the results by mutual consent to make one classification. Here, the inspectors include one Ph.D. student (first author), who worked at a software company for two years as a full-time developer, and one professor who has been studying FLOSS development over ten years.

4 Interview and Classification Results

4.1 Developer Demography (Q1)

As we described earlier, we invited 22,228 developers to join our interview. During the two weeks interview period, we got responses from 322 developers. Table 1 shows the product domains where the developers participated. We can see "web application" is the most popular domain (7%) but it does not stand out from the others. The product domains spread across a wide area. We can assume that the results of our interviews reflect a wide range of situations across FLOSS development. Table 2 shows the developers' experience with FLOSS development. The majority of the developers have over five years experiences. It is no surprise because we only invite active developers who have made at least over 100 commits to GitHub repositories. Table 3 shows developers' motivations to FLOSS development. 59% (126 + 64) of the developers contribute to FLOSS projects as part of work.

Table 1. Product domains where GitHub developers join (Q1-1)

Domain	#	%	Domain	#	%	Domain	#	%
Web application	22	7%	Database	9	3%	Machine learning	5	2%
Development tool	19	6%	Network server	9	3%	UI	5	2%
Analysis	19	6%	Messaging tool	9	3%	Mobile app	5	2%
Language & compiler	17	5%	Education	8	2%	Desktop system	4	1%
OS	15	5%	Simulator	7	2%	Mail	4	1%
Graphic	14	4%	Finance	7	2%	Browser	3	1%
Game	13	4%	Resource monitoring	7	2%	Others	37	11%
Programming tool	12	4%	Image editor	6	2%	No answer	34	11%
Blog	11	3%	Network tool	6	2%			
Embedded OS	9	3%	Security tool	6	2%	**Total**	**322**	**100**

Table 2. Experience with FLOSS development (Q1-2)

Experience	Developers
More than five years	213
Three to five years	63
One to three years	45
Less than one year	1

Table 3. Motivation to participate in FLOSS development (Q1-3)

Motivation	Developers
Hobby	111
Work	126
Both	64
Other	21

Table 4. A distribution of high impact bugs in Q2-1

High impact bugs	#	%
Security bug	171	53%
Breakage bug	72	22%
Performance bug	20	6%
Blocking bug	16	5%
Dormant bug	12	4%
Surprise bug	7	2%
Others	24	7%

4.2 RQ1: What Kinds of High Impact Bugs Are Mainly Considered High Impact by FLOSS Developers?

In Q2-1, we asked the developers to select one of the six kinds of high impact bugs which are introduced in the previous section and have been well studied in the SE community. Table 4 shows the responses from the developers. We can see the FLOSS developers from GitHub attach greater importance on security bugs (53%) and breakage bugs (22%). It was unexpected for us that the other four bugs attract less attention from the FLOSS developers. It partly indicates the perception of gaps between researchers and FLOSS developers. Some researchers in the SE community might misunderstand FLOSS developers' actual needs.

Researchers in the SE community have been studying to help developers find and fix bugs especially in terms of impacts on users' satisfaction and during the development process (release) [13]. Although high impact bugs have been studied individually so far, it was not clear if FLOSS developers are mostly concerned with particular high impact bugs. From the result of Q2-1, we now answer RQ1 as follows.

Answer to RQ1: Researchers have been dedicating to provide a means to find and fix a variety of high impact bugs, but FLOSS developers mainly emphasize a focus on security and breakage bugs.

4.3 RQ2: What Kinds of High Impact Bugs Do FLOSS Developers Encounter Most Frequently?

In Q2-2, we asked the developers to describe the high impact bugs they have encountered in the past. Many of the developers described characteristics of high impact bugs in the free text format and also gave us direct links to actual bug reports which present symptoms discussed among developers and users.

Table 5 shows symptoms of bugs considered high impact by the respondents. In the table, we count multiple times if a developer described several high impact bugs. The percentage in the table is the ratio of developers' answers in each category, but the total percentage does not become 100% due to the above reason. As we described earlier, we manually inspected and categorized the information on high impact bugs by symptom. In what follows, we summarize the classification result.

We had 249 valid answers from 192 developers about symptoms of high impact bugs which actually get FLOSS developers in trouble in the past. In Table 5, the most common symptom was "unexpected processing" responded by 17% of developers (42 cases). As regards "unexpected processing", we could confirm less in common with bug reports. They ranged from different calculation results to unexpected rendering. The next most frequent was "sudden stop" responded by 16% of developers (39 cases). The corresponding bug reports shown by the developers suggested us that it often happened due to null pointer exception, run-time error exception, segmentation error, and overflow. Although the

Table 5. Symptoms described in actual bug reports

Category	Subcategory	Description	#	%	Ref
Behavior	Disable start	Users cannot install, compile or start an application	19	**8%**	[1,27,36]
	Never start function	A function never start once a user clicks a button	21	**8%**	[5,30,32,34]
	Sudden stop	A program suddenly stops during running	39	**16%**	[2,25,26,33]
	Unexpected processing	A program does not output or behave as developers expected	42	**17%**	[31,32]
	Never finishing state	A process never finish (e.g, hang up and infinite loop)	5	2%	
	Unable to login	Users cannot login a system	4	2%	
	Others	Lack of items in display, wrong warnings, lower user experiences etc.	8	3%	
Effect	Lower performance	A program lowers performance (e.g, too large memory usage)	13	**5%**	[20–22,35]
	Damage other systems	A program damages other systems (e.g, OS cannot boot)	5	2%	
	Others	Making a disk full etc.	3	1%	
Security	Vulnerability	Security defects allow an attack to cause an abnormal behavior	22	**9%**	[6,17,29]
	Unauthorized access	An impersonating account accesses to a server, service, or data	28	**11%**	[12,16,19]
	DDoS	Massive accesses from multiple terminals make a service unable	7	3%	
Data	Data loss	A program deletes data in a product (e.g., user data and database breakage)	12	**5%**	
	Incorrect data	A program produces incorrect or duplicated data	1	0%	
Development	Architecture change	It forces developers to change a architecture or core program in a product	3	1%	
	Reproduce	Developers cannot reproduce a reported bug	3	1%	
	Others	Blocking other bugs fixed etc.	4	2%	
Reputation	Compatibility	Compatibility is broken (e.g, API, hardware and OS)	7	3%	
	Execution env.	A product can not guarantee an execution environment	3	1%	

above two are related to “Behavior” of a program, the third and fourth most common symptoms were “Security” concerns such as “vulnerability” and “unauthorized access.” About “vulnerability,” the corresponding bug reports suggested the developers especially concerned with XSS and SQL injection attacks. The OpenSSL problem (i.e., Heartbleed) and the hidden way of leaking user data were included in bug reports about “unauthorized access.”

In RQ1, 53% of developers think that security is the biggest concern among high impact bugs in the previous studies. However, the result in RQ2 shows that the developers come across high impact bugs about Behaviour more often than the one about security. In fact, one developer said, *“Since the mentioned project is (mostly) a client-side javascript library, security problems aren’t common.”* Based on the results here, we answer RQ2 as follows.

Answer to RQ2: FLOSS developers most frequently encounter bugs relating to behaviors such as unexpected behaviors and sudden stops. Security bugs such as vulnerabilities and unauthorized accesses are also often encountered.

4.4 RQ3: What Kinds of High Impact Bugs Should Be Studied Newly by the SE Community in Order to Support FLOSS Developers?

The percentages in Table 5 are indicated by boldface if the corresponding symptoms account for about 80% of all the symptoms (i.e., the developers frequently encounter the symptoms with boldface.). For the majority of the symptoms, we surveyed existing studies which have tried to find and/or resolve the symptoms and showed references as "Ref" in Table 5.

The percentages of the symptoms in "Behavior" category are relatively high and these have been well-studied by the SE community [1,2,5,25–27,30–34,36]. For instance, "never start function" is well studied *breakage bugs* [30] so-called regressions which disable existing functions due to additional changes to software products. Although in the paper we did not introduce this as a high impact bug, "unexpected processing" is well studied as a functionality bug or feature bug [32]. "disable start" and "sudden stop" are also studied as build bugs [36] and crash bugs [2] respectively.

As we confirmed "vulnerability" and "unauthorized access" achieved relatively high attention from the developers, *security bugs* are also considered high impact by researchers and have been well studied [6,12,16,17,19,29,35]. "Lower performance" in "Effect" category is also well studied [20–22,35] as performance bugs. However, to the best of our knowledge, there is no study on "data loss" in "Data" category which is of relatively high concern to FLOSS developers (5%). For instance, a bug on "data loss" in "Data" category is observed when deleting data related to the operation under a condition. Other data loss bugs occurred due to executing the wrong processing of multi-transaction or by using variables not multi-threaded (e.g., HashMap in Java). In fact, for instance, loss of users' data such as their pictures was recently reported in the update of Windows 10 [3]. We regard it is one of the perception gaps between FLOSS developers and SE researchers and should be should be studied, allowing us to address the issue. Based on the results here, we answer RQ3 as follows.

Answer to RQ3: Existing studies can cover FLOSS developers' concerns about high impact bugs, but researchers still have space to further study other kinds of high impact bugs such as "data loss".

5 Discussions

5.1 Is the Current Support for High Impact Bugs Enough? How Can We Help FLOSS Developers Fix Bugs?

Many studies focus on symptoms of bugs and observe their characteristics and impacts [4,7,22,30]. We classified bugs based on the symptoms of high impact bugs included in the answers from the interviews. However, it is not enough to support fixing bugs because we can not fix them only by knowing the symptoms of high impact bugs. In this section, similar to the classification of the symptoms, we classify causes of high impact bugs obtained in the answers from the interviews. With the classification of the causes, we discuss how we can support fixing the high impact bugs in each category.

Table 6 shows the causes of bugs considered high impact by the respondents. Here, we had 182 valid answers including the root causes of high impact bugs (from 142 developers). In Table 6, the most common root cause was "insufficient processing" reported by 35% developers (49 cases). Furthermore, we broke down the 49 cases of "insufficient processing" and found that they consisted of 13 cases of "insufficient checks for inputs by users", nine cases of "insufficient checks against malicious inputs", five cases of "insufficient null checks such as arguments and return values", four cases of "insufficient authority checks "(e.g., database), and "others" (16 cases). The second and third most common root causes were "problem in 3rd party" and "change in 3rd party" respectively which relate to 3rd party libraries and systems.

Figure 1 depicts the relationship between symptoms and causes. The line width shows how symptoms and causes have strong relationships. For ease of reading graph, we only show the relationships that appeared more than two-times (which account for 52% of all the relationship). For further information, we provide the data and the figures showing all the relationships on our online appendix [24]. Looking into the causes of "unexpected processing", the most frequent causes are "insufficient processing" (11 cases), and "problem in 3rd party" (5 cases), and "usage" (4 cases) are following. Furthermore, we investigated the detail of the bugs and found that nine of 11 cases were happened via regular usages and two cases were occurred by abnormal usage. These bugs should be found by unit test, or developers can utilize many studies (e.g, test case generation) to find them. "Sudden stop" is also caused by "insufficient processing" (11 cases). Eight of the 11 cases are caused by insufficient checks for inputs by users or by insufficient null checks, which are able to be applied by binary analysis [25].

The causes of both "disable start" and "never start function" are related on 3rd parties. The main causes of "disable start" are "changes in 3rd party". We found two interesting answers indicating difficulty in finding the bugs are brought by "changes in 3rd party". One developer said, *"It was just to update dependencies but finding bug was hard"*. Moreover, another developer said, *"Because there are too many packages in the repository, nobody can keep an eye on all the packages"*. Even though developers roughly know that the causes of the bugs are caused by changes in 3rd parties, it is difficult to specify which changes by third

Table 6. Root causes described in actual bug reports

Category	Subcategory	Description	#	%
Design	Usage	A program usage unanticipated by developers (e.g., no network)	12	7%
	Architecture	An architecture has an inappropriate algorithm etc.	2	1%
	Wrong processing	A program has incorrect processing	10	5%
	Superfluous processing	A program have superfluous processing	3	2%
	Insufficient processing	A program needs another processing (e.g., null checks)	39	21%
	Others	Incorrect encryption etc.	6	3%
Implement	Exception handling	A program throws a wrong Exception or cannot catch it	11	6%
	Condition	A program has wrong conditions (e.g., for and if)	7	4%
	Usage of variables	A program use wrong variables or data type	5	3%
	Method use	Developers incorrectly use methods in their product or 3rd party library	8	4%
	Memory management	A program cannot appropriately manage memory	9	5%
	Others	Wrong implementation of async, data or race competition etc.	6	3%
Operation and Maintenance	Problem in 3rd party	A program is affected on a problem in 3rd party library or systems	21	12%
	Changes in 3rd party	A program is affected on a change in 3rd party library or systems	13	7%
	Execution env.	Developers did not check if a program can run on some OS or shells	9	5%
	Change effects	A change in the module affects on other modules	5	3%
	Setting	Developers use a wrong setting (e.g, buffer size and database access authority)	5	3%
	Others	Incorrect document, refactoring, or operation etc.	11	6%

party products created the bug. Although Ma [15] et al. investigated common practice to fix cross-project correlated bugs in the GitHub scientific Python ecosystem, there are no approaches to specify the root causes of cross-project bugs.

“Vulnerability” and “unauthorized access” are mostly caused by “insufficient processing” (6 cases each). Among the subcategories of “insufficient processing”, the most common causes are derived from insufficient checks against malicious inputs (3 cases each). Fuzzing techniques [11,14] are able to be used to find the insufficient checks.

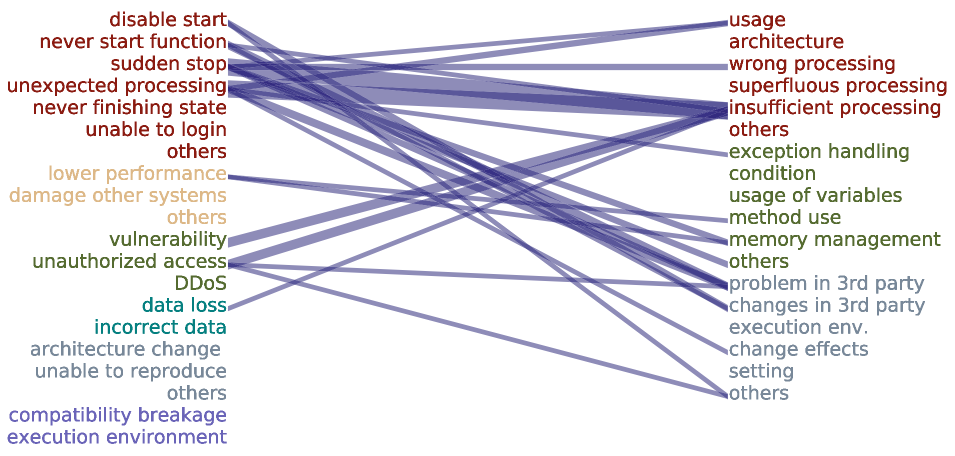

Fig. 1. Relationships between symptoms (left axis) and causes (right axis)

The cause of "data loss" is "insufficient processing", which can be broken down into three different cases: "insufficient checks for input by users", "insufficient checks against malicious inputs", and "others". Because of no similarities, we could not find any approaches to address "data loss" bugs, therefore it requires further work in the future.

5.2 Threats to Validity

Internal validity: The categorization of Table 5 and 6 may not be perfect. We have a good deal of knowledge about software, but we also recognize the limitations of our knowledge about specific domains. We also might bias in creating the category. **External validity:** Although the developer demography consists of developers working in a wide range of product domains, a judgment if a bug is high impact or not would depend on a product domain. **Construct validity:** To avoid bias in the developers responses, we asked them about high impact bugs without providing rigorous definitions of high impact bugs. Attitudes towards high impact bugs might be different among the developers.

6 Conclusion and Future Work

In this study, we interviewed 322 notable GitHub developers to reveal the perception gaps between FLOSS developers and researchers on bug finding and fixing. We manually inspected and classified actual bug reports which were presented and considered high impact by the developers. As a result, we concluded that security and breakage bugs are highly important for FLOSS developers. We also identified "data loss" bugs should be studied newly by the SE community to support FLOSS developers. Based on the bug report data presented by the developers in this study, we plan to investigate actual impacts of bugs on the size of source code change, resolution time, and so forth in the future.

Acknowledgment. We really appreciate the cooperation of developers in GitHub in completing our survey. This research is conducted as part of Grant-in-Aid for Japan Society for the Promotion of Science Research Fellow and Scientific Research (JP17J03330, JP17H00731, JP18K11243).

References

1. Abate, P., Di Cosmo, R., Gesbert, L., Le Fessant, F., Treinen, R., Zacchiroli, S.: Mining component repositories for installability issues. In: Proceeding of the 12th Working Conference on Mining Software Repositories, pp. 24–33 (2015)
2. An, L., Khomh, F., Guéhéneuc, Y.G.: An empirical study of crash-inducing commits in Mozilla Firefox. Softw. Qual. J. **26**(2), 553–584 (2018)
3. Ars Technica. https://arstechnica.com/gadgets/2018/10/microsoft-suspends-distribution-of-latest-windows-10-update-over-data-loss-bug/
4. Chen, T.H., Nagappan, M., Shihab, E., Hassan, A.E.: An empirical study of dormant bugs. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 82–91 (2014)
5. Felsing, D., Grebing, S., Klebanov, V., Rümmer, P., Ulbrich, M.: Automating regression verification. In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering, pp. 349–360 (2014)
6. Gao, F., Wang, L., Li, X.: BovInspector: automatic inspection and repair of buffer overflow vulnerabilities. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 786–791 (2016)
7. Garcia, H.V., Shihab, E.: Characterizing and predicting blocking bugs in open source projects categories and subject descriptors. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 72–81 (2014)
8. Gegick, M., Rotella, P., Xie, T.: Identifying security bug reports via text mining: an industrial case study. In: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, pp. 11–20 (2010)
9. GitHub. https://github.com/
10. GitHub API. https://developer.github.com/v3/
11. Jiang, B., Liu, Y., Chan, W.K.: ContractFuzzer: fuzzing smart contracts for vulnerability detection. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 259–269 (2018)
12. Kafali, O., Jones, J., Petruso, M., Williams, L., Singh, M.P.: How good is a security policy against real breaches? A HIPAA case study. In: Proceedings of the 39th International Conference on Software Engineering, pp. 530–540 (2017)
13. Kashiwa, Y., Yoshiyuki, H., Kukita, Y., Ohira, M.: A pilot study of diversity in high impact bugs. In: Proceedings of the 30th International Conference on Software Maintenance and Evolution, pp. 536–540 (2014)
14. LibFuzzer. https://llvm.org/docs/LibFuzzer.html
15. Ma, W., Chen, L., Zhang, X., Zhou, Y., Xu, B.: How do developers fix cross-project correlated bugs? A case study on the GitHub scientific python ecosystem. In: Proceedings of IEEE/ACM 39th International Conference on Software Engineering, pp. 381–392 (2017)
16. Mathis, B., Avdiienko, V., Soremekun, E.O., Bohme, M., Zeller, A.: Detecting information flow by mutating input data. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 263–273 (2017)

17. Meng, N., Nagy, S., Yao, D., Zhuang, W., Argoty, G.A.: Secure coding practices in Java: challenges and vulnerabilities. In: Proceedings of the 40th International Conference on Software Engineering, pp. 372–383 (2018)
18. Molyneaux, I.: The Art of Application Performance Testing: Help for Programmers and Quality Assurance, 1st edn. O'Reilly Media Inc., Newton (2009)
19. Near, J.P., Jackson, D.: Finding security bugs in web applications using a catalog of access control patterns. In: Proceedings of the 38th International Conference on Software Engineering, pp. 947–958 (2016)
20. Nguyen, T.H.D., Nagappan, M., Hassan, A.E., Nasser, M., Flora, P.: An industrial case study of automatically identifying performance regression-causes. In: Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 232–241 (2014)
21. Nistor, A., Chang, P.C., Radoi, C., Lu, S.: CARAMEL: detecting and fixing performance problems that have non-intrusive fixes. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 902–912 (2015)
22. Nistor, A., Jiang, T., Tan, L.: Discovering, reporting, and fixing performance bugs. In: Proceedings of the 10th Working Conference on Mining Software Repositories, pp. 237–246 (2013)
23. Nurmuliani, N., Zowghi, D., Williams, S.: Using card sorting technique to classify requirements change. In: Proceedings of 12th International Requirements Engineering Conference, pp. 224–232 (2014)
24. Online_Appendix. https://github.com/YutaroKashiwa/OSS2019
25. Pham, V.T., Ng, W.B., Rubinov, K., Roychoudhury, A.: Hercules: reproducing crashes in real-world application binaries. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 891–901 (2015)
26. Seo, H., Kim, S.: Predicting recurring crash stacks. In: Proceedings of the 27th International Conference on Automated Software Engineering, p. 180 (2012)
27. Seo, H., Sadowski, C., Elbaum, S., Aftandilian, E., Bowdidge, R.: Programmers' build errors: a case study (at Google). In: Proceedings of the 36th International Conference on Software Engineering, pp. 724–734 (2014)
28. Shafiq, H., Arshad, Z.: Automated debugging and bug fixing solutions: a systematic literature review and classification, M.Sc thesis, Blekinge Institute of Technology (2014)
29. Shar, L.K., Beng Kuan Tan, H., Briand, L.C.: Mining SQL injection and cross site scripting vulnerabilities using hybrid program analysis. In: Proceedings of the 35th International Conference on Software Engineering, pp. 642–651 (2013)
30. Shihab, E., Mockus, A., Kamei, Y., Adams, B., Hassan, A.E.: High-impact defects: a study of breakage and surprise defects. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 300–310 (2011)
31. Sullivan, M., Chillarege, R.: Software defects and their impact on system availability-a study of field failures in operating systems. In: Proceedings of the Fault-Tolerant Computing: The Twenty-First International Symposium, pp. 2–9 (1991)
32. Tan, L., Liu, C., Li, Z., Wang, X., Zhou, Y., Zhai, C.: Bug characteristics in open source software. Empir. Softw. Eng. **19**(6), 1665–1705 (2014)
33. Tan, S.H., Dong, Z., Gao, X., Roychoudhury, A.: Repairing crashes in Android apps. In: Proceedings of the 40th International Conference on Software Engineering, pp. 187–198 (2018)

34. Tan, S.H., Roychoudhury, A.: Relifix: automated repair of software regressions. In: Proceedings of the 37th International Conference on Software Engineering, vol. 1, pp. 471–482 (2015)
35. Zaman, S., Adams, B., Hassan, A.E.: Security versus performance bugs: a case study on Firefox. In: Proceedings of the 8th Working Conference on Mining Software Repositories, pp. 93–102 (2011)
36. Zhao, X., Xia, X., Kochhar, P.S., Lo, D., Li, S.: An empirical study of bugs in build process. In: Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1187–1189 (2014)