Build Issue Resolution from the Perspective of Non-Contributors

Sunzhou Huang
sunzhou.huang@utsa.edu
University of Texas at San Antonio
San Antonio, Texas, USA

Xiaoyin Wang
xiaoyin.wang@utsa.edu
University of Texas at San Antonio
San Antonio, Texas, USA

Abstract

Open-source software (OSS) often needs to be built by roles who are not contributors. Despite the prevalence of build issues experienced by non-contributors, there is a lack of studies on this topic. This paper presents a study aimed at understanding the symptoms and causes of build issues experienced by non-contributors. The findings highlight certain build issues that are challenging to resolve and underscore the importance of understanding non-contributors' behavior. This work lays the foundation for further research aimed at enhancing the non-contributors' experience in dealing with build issues.

CCS Concepts

• **Information systems** → **Open source software**; • **Software and its engineering** → **Application specific development environments**.

Keywords

open source software, development environment, build issue resolution

ACM Reference Format:
Sunzhou Huang and Xiaoyin Wang. 2024. Build Issue Resolution from the Perspective of Non-Contributors. In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3691620.3695304

1 Introduction

Open-source software (OSS) frequently requires building from source code. This "build" process often encompasses several steps, including compiling the source code, resolving dependencies, packaging, testing, performing static analysis, generating documentation, and preparing for deployment. The primary roles of those engaged in these building activities are those of project contributors, who manage repositories and commit code. However, there are also numerous non-contributors who need to build the project for their own goals. Examples of such non-contributors are as follows:

Users: Not all OSS projects provide pre-built installation packages. Even when available, these packages may not be compatible with a user's operating system (OS) or local environment. So some users may need to build projects from source code in their local environments. Advanced users might also prefer non-default configurations, utilize new features not yet included in a stable release, or compile directly from the source code for security reasons, such as in blockchain scenarios [1].

Learners: Individuals who are new to OSS development and are in the learning phase also engage in building OSS. They may start by compiling and experimenting with the source code to understand the project structure and functionality better. This hands-on experience is essential for their growth as OSS contributors.

Potential contributors: Individuals who wish to propose feature improvements or bug fixes for OSS must also build the project initially. This step is necessary before they can validate their revisions and submit a pull request. However, because OSS often has diverse build environments, these potential contributors frequently lack the required local build setups. Setting up these environments themselves can pose its own set of challenges.

In this paper, within the context of the OSS build process, we use the term "non-contributor" to refer to roles that lack familiarity with the build environment of the target OSS and need to set up their local environment. These roles may have varying degrees of background knowledge about the target OSS, but they all share the common objective of building the OSS from its source code.

Since non-contributors are often unfamiliar with the build environments of the OSS projects they aim to work on, many encounter challenges in this process. On Stack Overflow (SO), filtering by the keywords "not build" yielded 6,760 out of 43,140 (15.7%), 7,374 out of 95,275 (7.73%), and 10,336 out of 73,559 (14.1%) relevant SO questions for three well-known open-source projects: Tomcat, Hibernate, and OpenCV, respectively [12]. Since OSS contributors are unlikely to address internal build issues on Stack Overflow (preferring internal issue tracking systems like GitHub Issues or JIRA), it is reasonable to infer that most of these SO questions are from non-contributors experiencing build issues in their local environments. These numbers highlight the prevalence and difficulty of build issues faced by non-contributors. Another illustrative example is evident in computer science (CS) classes, where students frequently encounter initial project hurdles. These difficulties typically arise from their inability to build or install the frameworks or tools on their own systems.

Despite the prevalence of build issues from non-contributors, existing research has largely overlooked these challenges, focusing primarily on the perspective of contributors, such as the developers of the project being built. For instance, extensive studies have been conducted on the effort required to maintain build scripts [10, 11], the categorization and distribution of bugs in build scripts [3, 16, 22], the patterns of fixing build scripts [9, 23], and how build failures occur in new integration scenarios such as Continuous Integration (CI) chains [24] or Docker environments [21]. It is important to note that build issues experienced by non-contributors may have

ASE '24, October 27-November 1, 2024, Sacramento, CA, USA

ACM ISBN 979-8-4007-1248-7/24/10
https://doi.org/10.1145/3691620.3695304

different root causes compared to those experienced by contributors. The latter are often caused by flaws in the build scripts or code of the project being built, while the former are likely related to the local environments of the non-contributors, assuming that most OSS projects are released with validated build scripts. One potential reason why build issues experienced by non-contributors are not studied may be due to data collection challenges. Most existing studies gather data from version histories and build logs, while non-contributors' experienced build issues are never systematically tracked or recorded.

In this paper, we present a study to investigate build issues experienced by non-contributors, aiming to answer the prevalent question, **Why does the OSS project not build on my machine?**. Our study tracks the behaviors of senior-year and graduate computer science students as they undertake build tasks for various OSS projects. We believe these students are representative subjects for our study, as they currently occupy learner roles for the target OSS projects and could potentially transition to user or potential contributor roles with appropriate guidance. All these students have a certain level of background in information technology (most have some internship and working experience), which positions them for a more advanced user role in utilizing the target OSS projects. We have designed these build tasks as a course project for a cross-listed software engineering course, where we instructed 31 students to build 12 different OSS projects in 6 programming languages (PLs) in order to explore the symptoms and causes of build issues experienced by non-contributors.

2 Related Works

The research efforts most related to our research are user studies on software build tasks. Kwan et al. [7] studied developers from IBM to find out whether team composition and coordination may have an impact on software build success. Dawns et al. [4] studied the operation of a build team to evaluate an automatic build management tool that enforces the build failure handling process. Philips et al. [13] studied software building teams at Microsoft and found that most challenges are on the social aspects of the team. Kerzazi et al. [6] studied 3,214 software builds and found that 17.9% of builds failed and more than 300 man-hours were cost to fix them. Hilton et al. [5] performed interviews with 16 developers to find out their opinion on whether the CI process may enhance software productivity. Vassallo et al. [19] performed a study with 17 participants to find out how well developers can take advantage of BART, a tool for summarizing build failure reasons. Different from our research, all these studies are from the perspective of project managers or senior developers instead of non-contributors.

Besides user studies, there have been a lot of empirical studies on software build history and build failures. Mcintosh et al. studied the version histories of proprietary and open source software projects to estimate the effort required to repair build scripts [10] and to correlate effort with type of bulid systems [11]. Xia et al. [22] performed studies to summarize bugs in software build systems. Zhao et al. [23] studied build failure reports in five OSS and found that build failures take much more time to fix than others. Barrak et al. [3] studied how build failures are correlated with code smells in build scripts and code. Licker and Rice [8] investigated the incorrect rules in build scripts by using a mutation testing approach. Wu et al. [21] studied the characteristics of build failures in the context of docker environments. These existing studies focus on build failure logs and build failure fixes in the commit history. In contrast to previous studies, our research collects and analyzes the entire process of completing multiple OSS build tasks in a local environment, taking into account the system environment factors that influence the build process.

3 Study Design

We conducted a study that gathered data on build issues from 31 participants involved in 12 OSS projects. This provided us with a comprehensive understanding, enabling us to identify key symptoms and examine their resolution process. We aim to answer the following research question:

What are the common symptoms of build issues experienced by non-contributors during the build process, and to what extent can these symptoms be mitigated?

3.1 Participants and Tasks

Our participants were 31 students enrolled in the same software testing course. This course is a cross-listed elective for senior-year undergraduates and graduate students, with software engineering as a prerequisite. The course, taught by two of the authors at a university, focuses on software testing approaches, test planning, test case design, and build systems with CI/CD concepts. In our study, students shared many characteristics with non-contributors who only needed to build the released OSS without modifying the source code. None of the students have the experience to set up the specific build environments required by the OSS projects in our study. Almost all of the students will work or are already working part-time or full-time as developers, so studying their behavior could further help us understand the build issues when newcomers are onboarding.

As shown in Table 1, we selected 6 popular PLs from the top 15 used on GitHub. For each PL, we chose 2 of the top 10 most popular OSS projects on GitHub, ranked by the number of stars. We made our best effort to choose PLs and OSS projects with distinct real-world application scenarios. Participants picked a project from the 12 OSS projects on a first-come, first-served basis. We limited each project to three slots to ensure that each project had at least one participant. Participants were instructed to write a report documenting at least 10 non-trivial build issues. They were encouraged to resolve the issues if they could.

Table 1: Selected OSS projects

PL	Project	No. Participants	No. Issues
C++	opencv/opencv	3	30
	tensorflow/tensorflow	3	38
Go	gohugoio/hugo	4	32
	kubernetes/kubernetes	2	18
Java	elastic/elasticsearch	2	18
	spring-projects/spring-boot	3	46
JavaScript	electron/electron	1	4
	vuejs/vue	4	36
PHP	fzaninotto/Faker	2	9
	guzzle/guzzle	1	15
Python	pallets/flask	3	26
	scikit-learn/scikit-learn	3	31
Total		**31**	**303**

Students were required to complete the study as one of their major course projects. We designed a three-stage task list to help participants become familiar with the study process: 1) Setup. Participants were instructed to use the Google Cloud Platform (GCP) online console to set up a GCP project with $50 education credits and enabled APIs for creating virtual machines (VMs). 2) Warm-up. Participants were instructed to create a warm-up VM instance. They then performed warm-up build activities to familiarize themselves with the logging and snapshot-taking processes. 3) Build OSS projects. Each of the participants picked one OSS project for their build tasks. They utilized log scripts, snapshots, and textual issue reports to document build issues encountered during the build process of the target OSS project.

The anticipated outcomes of the study included successfully compiling the OSS projects and passing all tests using the commands specified in the build tasks. Beyond the build outcome, participants' grades would be evaluated based on their effort invested in completing the tasks, as evidenced by snapshots and issue reports. This grading approach was designed to accommodate participants who might not be able to successfully build the OSS projects. As the conductors of the study, our role was to provide clarity on the objectives of the build tasks. Our study protocol underwent assessment and received approval from our local Institutional Review Board (IRB). All students were informed that they may withdraw their data after grading, ensuring their data would not be included in our dataset for this study or any future research activities.

3.2 Data Collection

We designed protocols to collect data throughout the build process. As participants performed build tasks on a virtual machine (VM), we used the following methods to capture all relevant data.

VM state logging. When encountering build issues, participants used a script to log command history, environment variables, and network information. This logging helped correlate system data with reported issues. Participants also logged the final VM state upon completing tasks, with all logs saved for later analysis.

VM snapshots. Snapshots captured the entire VM state at specific moments, including files, settings, and configurations. Participants manually took snapshots upon encountering build issues and upon task completion, though temporary environment variables could be lost in the process.

Build issue report. Participants documented build issues, including inputs, outputs, and solutions, along with details about the build environment (e.g., language versions, tools). They summarized their process and provided feedback, offering insights into their problem-solving strategies.

4 Results and Analysis

We collected a total of 303 build issues from 31 build issue reports, along with 380 snapshots containing environment logs. One student changed from the "electron/electron" project to "vuejs/vue" due to low GCP credit. One student worked on the wrong project in Go. Eventually, for each OSS, there was at least one student and a maximum of four students. For each PL, there were at least three students and a maximum of six students. In the collected 303 build issues, the minimum number of issues reported was 4 for the "electron/electron" project, with only 1 student participant. The maximum number of issues reported was 46 for the sprint-boot project, with 3 participants. The number of reported issues was not correlated with only the number of participants but was also project-specific.

We implemented a qualitative analysis on these 303 build issues, utilizing the open coding procedure [15]. This procedure was executed by two of the authors. The first author coded all build issues, drawing from error messages and solutions reported within issue reports, and identified any borderline cases. Subsequently, the second author validated this coding and engaged in discussions about any borderline cases. In instances where conflicts arose, the first author leveraged corresponding snapshots and collected environment information to attempt to reproduce the build issue. Decisions were made based on the reproduced build results or the history available in the snapshot, with the two authors discussing and reaching consensus. Additionally, two authors were responsible for labeling the resolution state of the issues. If no solution was provided, the issue was labeled as *Unsettled*. However, if either of the two authors believed that a solution or workaround was provided, the issue was labeled as *Settled*.

Figure 1 illustrates the 14 distinct symptoms of build issues identified in our study. The light-colored bars represent unsettled build issues, while the remaining dark-colored bars denote settled issues. Out of the total 303 issues analyzed, 182 (60.1%) remained unsettled, highlighting the persistent challenges faced in addressing build issues experienced by non-contributors. To better understand the characteristics of these symptoms, we further categorized them into four broad categories: environment, process, system, and test-related issues. By grouping the 14 symptoms into these categories, we aim to provide a structured analysis of the underlying factors contributing to build issues. In the following sections, we examine each category, analyzing the symptoms and their implications for build issue resolution.

Environment-related symptoms include those related to incompatibility and missing components, which together account for the majority (181 out of 303) of build issues. Incompatibility-related symptoms, such as those involving build tools, dependencies (Dep.), and PL, occur when OSS projects cannot work with necessary components. Missing-related symptoms arise when build tools, dependencies, environment (Env.) variables (Var.), PLs, and external resources are not available in the current workspace. These two dependency-related symptoms have a high unsettled rate, a finding that aligns with studies of contributors-experienced build

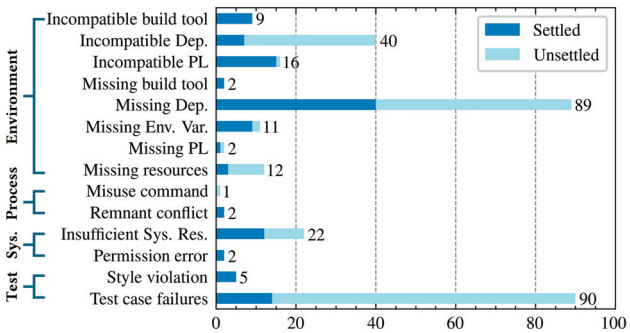

Figure 1: Symptoms experienced by non-contributors.

issues [9, 18, 17, 25]. Missing resources, common in evolving OSS systems, are challenging for non-contributors to resolve due to the difficulty in fetching external resources. The incompatible PL has a good settlement rate, primarily because the error messages in selected projects are easy to identify, unlike those for incompatible dependencies. Missing environment variables may mask other symptoms, as those issues require specific environment variables. We observed during the coding process that many incompatibility symptoms can be converted to other symptoms as non-contributors seek solutions.

Process-related symptoms refer to build issues caused by participants' mistakes during key steps of the build process. For example, "misuse command" symptom can occur when participants fail to escape special characters in command options, and "remnant conflict" symptom can arise when participants neglect to clean up the build remnants. Resolving these issues can be complex if participants are unable to correctly interpret the error messages, and the difficulty can be compounded by the complexity of the required build process.

System-related symptoms refer to "Insufficient system (Sys.) resources (Res.)" and "Permission error". The insufficient system resources symptom ranks among the top four symptoms, as most of the selected OSS projects are real-world projects that require a larger amount of system resources than our participants anticipated. Moreover, non-contributors often lack an effective approach to estimating the system resources required before they perform the build activities. This could explain the high unsettled rate of 10 out of 22. This symptom is also common since VMs created by our participants typically do not have many redundant system resources. For instance, participants experienced system crashes due to disk space exhaustion, followed by failures to reconnect to VMs. They did not realize the issue until the system resources were exhausted. Another symptom is the "permission error", which is caused when underlying software requires higher permissions than it currently has. In our study, we found that *Docker* commands often need superuser permissions. This is a common issue among OSS projects that rely on *Docker* [2].

Test-related symptoms refer to "Style violations" and "Test case failures". Style violation test failures typically occur when a project does not adhere to a certain standard. Participants can often find a workaround to bypass this issue. However, it is important to note that the root cause of this symptom may be due to compatibility issues. The most common symptom in our study is the "Test case failures" symptom, which also has a high unsettled rate. This aligns with existing studies [14, 20]. Notably, some test failure messages are difficult to interpret, making it hard to identify the root cause. The symptom could be the result of other underlying issues.

The distribution of these categories among the participants is as follows: "Environment" was experienced by 29 participants (93.6%), "Process" by 2 participants (6.5%), "System" by 10 participants (32.3%), and "Test" by 24 participants (77.4%). Figure 2 illustrates the distribution of categories across different PLs. Environment and test related symptoms are present in all PLs. In contrast, system-related symptoms are project-specific, which can be attributed to the fact that lightweight projects do not consume significant system resources. Upon further investigation of process-related symptoms, we observed that a participant encountered both

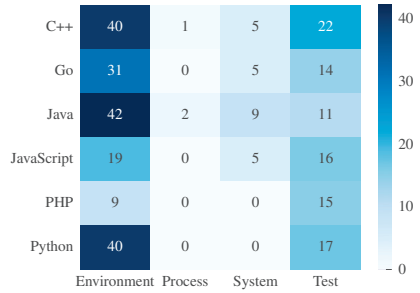

Figure 2: Distribution of categories on PLs.

"Misuse command" and "Remnant conflict" issues in a Spring Boot project. Despite its infrequent occurrence among the issues observed, the "Remnant conflict" issue was identified in two projects, one written in Java and the other in C++.

Compared to the study by Lou et al. [9], our study on the symptoms of build issues shows that non-contributors still frequently experience the same symptoms as contributors. However, these are mostly limited to issues caused by a lack of installation, version incompatibility errors, and environment variable issues. We also observed additional symptoms not covered by previous studies that are related to participants' actions, such as "Command misuse" and "remnant conflicts". As the 39.9% settlement rate shows, participants found it difficult to mitigate most of these frequent symptoms as well as identify the root causes of build issues. The strategies for resolving these build issues tend to be more straightforward due to the non-contributors' lack of specific project knowledge and experience. This limitation can make them more susceptible to encountering certain build issues.

5 Discussion and Future Work

Although containerization holds promise for addressing certain build issues, the lack of accessible container environments in non-contributors' local systems presented a challenge for many OSS projects, including most of those in our study. A threat to the validity of the findings is the subjectivity involved in determining whether a build issue is trivial. This judgment depends on several factors, including the individual's level of knowledge and local systems. Our future research will involve more participants and focus on a specific scope to explore the correlation between various factors and our findings. Additionally, we observed inconsistencies in reporting, such as misinterpreted test results and misaligned snapshots. Therefore, incorporating more monitoring measures into the process could significantly mitigate these potential threats.

6 Conclusion

This paper investigates 303 build issues experienced by 31 non-contributors. We found that non-contributors often struggle to resolve issues due to limited symptoms. Our study provides valuable insights into build issue resolution from the perspective of non-contributors, highlighting the importance of understanding their behavior and challenges. This work lays the foundation for further research in this area, with the ultimate goal of improving the experience of non-contributors in dealing build issues.[1]

[1]This work is supported in part by NSF 1736209 and 1846467.

References

[1] 2017. Difference between downloading bitcoire core from bitcoin.org and compiling from Github. https://bitcoin.stackexchange.com/questions/59875. Accessed: 2024-03-21.

[2] 2018. How to fix docker: Got permission denied issue. stackoverflow.com/questions/48957195. Accessed: 2024.

[3] Amine Barrak, Ellis E Eghan, Bram Adams, and Foutse Khomh. 2021. Why do builds fail?—A conceptual replication study. *Journal of Systems and Software* 177 (2021), 110939.

[4] John Downs, Beryl Plimmer, and John G Hosking. 2012. Ambient awareness of build status in collocated software teams. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 507–517.

[5] Michael Hilton, Nicholas Nelson, Danny Dig, Timothy Tunnell, Darko Marinov, et al. 2016. Continuous integration (CI) needs and wishes for developers of proprietary code. (2016).

[6] Noureddine Kerzazi, Foutse Khomh, and Bram Adams. 2014. Why do automated builds break? an empirical study. In *2014 IEEE International Conference on Software Maintenance and Evolution*. IEEE, 41–50.

[7] Irwin Kwan, Adrian Schroter, and Daniela Damian. 2011. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering* 37, 3 (2011), 307–324.

[8] Nándor Licker and Andrew Rice. 2019. Detecting incorrect build rules. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1234–1244.

[9] Yiling Lou, Zhenpeng Chen, Yanbin Cao, Dan Hao, and Lu Zhang. 2020. Understanding build issue resolution in practice: symptoms and fix patterns. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 617–628.

[10] Shane McIntosh, Bram Adams, Thanh HD Nguyen, Yasutaka Kamei, and Ahmed E Hassan. 2011. An empirical study of build maintenance effort. In *Proceedings of the 33rd international conference on software engineering*. 141–150.

[11] Shane McIntosh, Meiyappan Nagappan, Bram Adams, Audris Mockus, and Ahmed E Hassan. 2015. A large-scale empirical study of the relationship between build technology and build maintenance. *Empirical Software Engineering* 20 (2015), 1587–1633.

[12] Stack Overflow. 2024. Stack Overflow: a question-and-answer website for computer programmers. https://stackoverflow.com/

[13] Shaun Phillips, Thomas Zimmermann, and Christian Bird. 2014. Understanding and improving software build teams. In *Proceedings of the 36th international conference on software engineering*. 735–744.

[14] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 345–355. https://ieeexplore.ieee.org/abstract/document/7962384/

[15] C.B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on Software Engineering* 25, 4 (July 1999), 557–572. https://doi.org/10.1109/32.799955

[16] Mini Shridhar, Bram Adams, and Foutse Khomh. 2014. A qualitative analysis of software build system changes and build ownership styles. In *Proceedings of the 8th ACM/IEEE international symposium on empirical software engineering and measurement*. 1–10.

[17] Matúš Sulír and Jaroslav Porubän. 2016. A quantitative study of Java software buildability. In *Proceedings of the 7th International Workshop on Evaluation and Usability of Programming Languages and Tools*. ACM, Amsterdam Netherlands, 17–25. https://doi.org/10.1145/3001878.3001882

[18] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2017. There and back again: Can you compile that snapshot? *Journal of Software: Evolution and Process* 29, 4 (April 2017), e1838. https://doi.org/10.1002/smr.1838

[19] Carmine Vassallo, Sebastian Proksch, Timothy Zemp, and Harald C Gall. 2020. Every build you break: developer-oriented assistance for build failure resolution. *Empirical Software Engineering* 25 (2020), 2218–2257.

[20] Carmine Vassallo, Gerald Schermann, Fiorella Zampetti, Daniele Romano, Philipp Leitner, Andy Zaidman, Massimiliano Di Penta, and Sebastiano Panichella. 2017. A tale of CI build failures: An open source and a financial organization perspective. In *2017 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 183–193. https://ieeexplore.ieee.org/abstract/document/8094420/

[21] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang. 2020. An empirical study of build failures in the docker context. In *Proceedings of the 17th international conference on mining software repositories*. 76–80.

[22] Xin Xia, Xiaozhen Zhou, David Lo, Xiaoqiong Zhao, and Ye Wang. 2014. An empirical study of bugs in software build system. *IEICE TRANSACTIONS on Information and Systems* 97, 7 (2014), 1769–1780.

[23] Xiaoqiong Zhao, Xin Xia, Pavneet Singh Kochhar, David Lo, and Shanping Li. 2014. An empirical study of bugs in build process. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*. 1187–1189.

[24] Mahdis Zolfagharinia, Bram Adams, and Yann-Gaël Guéhénuc. 2017. Do not trust build results at face value-an empirical study of 30 million cpan builds. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 312–322.

[25] Mahdis Zolfagharinia, Bram Adams, and Yann-Gaël Guéhéneuc. 2019. A study of build inflation in 30 million CPAN builds on 13 Perl versions and 10 operating systems. *Empirical Software Engineering* 24, 6 (2019), 3933–3971.