The Impact of Failing, Flaky, and High Failure Tests on the Number of Crash Reports Associated with Firefox Builds

Md Tajmilur Rahman[*]
PBSC Urban Solutions
Longueuil, QC, Canada
trahman@pbsc.com

Peter C. Rigby
Department of Computer Science and Software Engineering, Concordia University
Montreal, QC, Canada
peter.rigby@concordia.ca

ABSTRACT

Testing is an integral part of release engineering and continuous integration. In theory, a failed test on a build indicates a problem that should be fixed and the build should not be released. In practice, tests decay and developers often release builds, ignoring failing tests. In this paper, we studying the link between builds with failing tests and the number of crash reports on the Firefox webbrowser. Builds with all tests passing have a median of only two crash reports. In contrast, builds with one or more failing tests are associated with a median of 508 and 291 crash reports for Beta and Production builds, respectively. We further investigate the impact of "flaky" tests, which can both pass and fail on the same build, and find that they have a median of 514 and 234 crash reports for Beta and Production builds. Finally, building on previous research that has shown that tests that have failed frequently in the past will fail frequently in the future, we find that Builds with *HighFailureTests* have a median of 585 and 780 crash reports for Beta and Production builds. Unlike other types of test failures, *HighFailureTests* have a larger impact on Production releases than on Beta builds, and they have a median of 2.7 times more crashes than builds with normal test failures. We conclude that ignoring test failures is related to a dramatic increase in the number of crashes reported by users.

CCS CONCEPTS

• **Software and its engineering → Software testing**;

KEYWORDS

Software Testing, User Crash Reports, Builds, Flaky Tests

ACM Reference Format:
Md Tajmilur Rahman and Peter C. Rigby. 2018. The Impact of Failing, Flaky, and High Failure Tests on the Number of Crash Reports Associated with Firefox Builds. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3236024.3275529

[*]This work was completed while Rahman was a PhD student at Concordia

1 INTRODUCTION

Software builds are tested to ensure that the functionality of the system is not broken by a change. Developers write test cases when they are developing new features or fixing bugs. In a rapid release model, fixed and shortened release schedules reduce the time for investigation of the test regressions [13]. We examine the impact of ignored failing tests, flaky tests, and *HighFailureTests* on the build quality as measured by the number of user crash reports associated with a build.

We organize our research around the following questions:

(1) **RQ1, Number of Crashes: How many crashes are there for builds on dev, beta, and production?**
Firefox stages its development into three channels. The development channel contains the current work being done by developers. The beta channel is used by early testers and users. The production channel is released to end users. The stability of the code and the number of users increase as we move from the Dev to Production channel. This first research question quantifies the number of crashes on each channel. This basic information is important to put the remaining research questions into context as low use channels will likely have few crashes but may not be of high quality.
(2) **RQ2, Test Failures: How many crashes are associated with builds that contain test failures?**
This research question quantifies the impact of test failures on crashes. Our goal is to understand if ignored test failures lead to an increase in end user crash reports.
(3) **RQ3, Flaky Tests: How many crashes are associated with builds that contain flaky tests?**
Flaky tests fail non-deterministically[12]. For example, a test may both pass and fail on the same build. As a result, developers cannot trust a flaky test to determine software quality. Our goal is to understand if ignored flaky test failures lead to an increase in the number of browser crashes.
(4) **RQ4, Historical Failures: Do failures of tests that have failed many times in the past lead to an increase in crashes?**
Researchers have shown the tests that have failed in the past tend to continue to fail at high levels [10]. These *HighFailureTests* allowed researches to re-order tests based on their historical likelihood to fail [1, 18]. We consider tests that historically fail 10% of the time to be *HighFailureTests*. We investigate whether failures of these tests lead to an increase in the number of crash reports.

The paper is organized as follows, Section 2 provides the background on the Firefox project's build process and crash collection. Section 3 describes the research methodology. This section also describes the data used in this case study. Section 4 discusses the results for each research question. Section 5 positions our work in the literature on build systems and testing. Section 6 concludes the paper.

2 FIREFOX RELEASE PROCESS

Firefox is a popular open source modern web browser and has been funded by the Mozilla Corporation since November 2004. Firefox's release process involves three channels: Development, Beta,[1] and Production [7–9]. Code remains on each channel for six weeks before transitioning to the next channel. Each channel has a different level of stability, purpose, and number of developers and users who exercise it. McIntosh *et al.* [15] found that the number of users per channel is 100K for Development, 1M for Beta, and 100M+ for the Release channel.

To create a release, a continuous integration tool, Buildbot, is used through `Bootstrap` automation scripts to build newly committed features into a new release [5]. The Buildbot master creates the build logs and manages the overall process. Each build has a report that contains the logs for each build and includes basic information about the build-setup, environment, test steps, the test verdict, and the overall build result.

When the browser closes unexpectedly a dialogue box allows users to submit crash reports [6]. Each submitted crash report contains a crash dump including the crashing page address, user's local environment, and the Firefox build id.

3 METHODOLOGY AND DATA

Our goal is to investigate the impact of ignored failing tests and flaky tests on the number of reported end user browser crashes. We follow a straight-forward method for our study. After loading the data into a database we normalize the test status into three categories: "Pass", "Fail" and "Flaky". We calculate which tests are historically *HighFailureTests*. We then link the build and crash reports based on the build id. We use R to provide statistical answers to our research questions. Figure 1 illustrates our research methodology.

3.1 Data

We collect the historical build logs and crash reports for Mozilla Firefox spanning from December 2010 to December 2012. We parse the build logs and store the extracted information in a database. The top portion of the log file contains the basic build summary including information about the builder, slave process, start time, pass or fail verdict, build id, and source code revision number (commit hash).

The test information is contained at the end of the build log file and includes the test status, test path, and a short description of the test. We use the test path to uniquely identify each test. The path is a URL that is linked to the test steps.

We then parse and extract all the crash reports into the database. Each crash report contains a crash signature, URL with an unique id, build id, operating system and other information that may be useful to developers. Once we load the data into the database, we remove incomplete data-rows that have missing information, such as the crashes with no build id.

After extracting the build logs and crash reports we have two data sets containing the crash information and the build history. The attributes are listed in Table 1. By joining the two data sets of builds and crashes we extract the builds that could be mapped with one or more crashes. For each build we extract the test steps from the build log and store them separately. We link them based on *build_id* and we found a total of 2.8K unique build ids that have both crash and test information. Associated with these builds are 729K crashes.

Table 1: Attributes for the build and crash data

Build Data Attr.	Crash Data Attr.
build_id	build_id
build_uid	url
revision	uuid_url
start_time	crash_date
test_info	signature
test_description	-
test_name	-

3.2 Test Status Mapping

We found six statuses that Firefox developers use to label their tests. Since there is no formal definition for these test labels, we examined the code of the test scripts [4] [3]. For this paper, we map the test statuses into three categories: Pass, Fail, and Flaky tests. The mapping between Firefox statuses and the categories is found in Table 2 along with the number of test runs associated with each status.

In Table 2 the status "PASS" maps to a normal test pass. The "FAIL", "UNEXPECTED-PASS", and "UNEXPECTED-FAIL" are categorized under the "Fail" category. In contrast, the "PASS(EXPECTED RANDOM)" and "KNOWN-FAIL(EXPECTED RANDOM)" are seen as failing and passing non-deterministically and we consider them to be flaky tests.

3.3 Identifying Flaky Tests

Flaky tests non-deterministically lead to a pass or fail verdict. Lou *et al.* identified flaky tests in their study [2] by searching for the keywords "intermittent" and "flak" within the commit history. They used commit logs for identifying flaky tests because they were mostly interested in flaky tests that are already fixed. However, we do not use a keyword search to identify flaky tests. We use the existing Firefox classification in the build log. In the test logs the tests that are marked as "*-RANDOM", we include them in the "Flaky" category which means, tests that are labelled with the statuses "PASS(EXPECTED RANDOM)" and "KNOWN-FAIL(EXPECTED RANDOM)" are considered to be the flaky tests (See table 2).

[1]Beta was original divided into two channels: Aurora and Beta

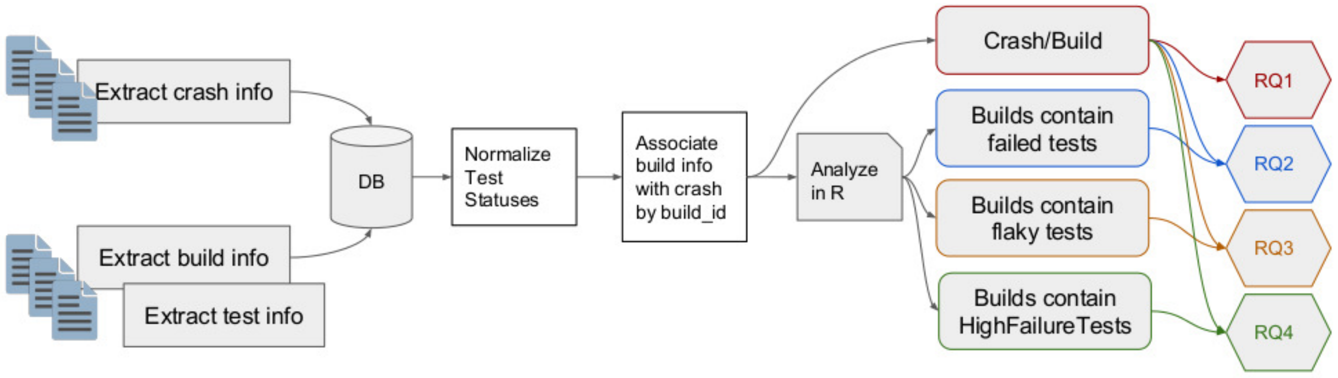
Extract crash info
Extract build info
Extract test info
DB
Normalize Test Statuses
Associate build info with crash by build_id
Analyze in R
Crash/Build
Builds contain failed tests
Builds contain flaky tests
Builds contain HighFailureTests
RQ1
RQ2
RQ3
RQ4

Figure 1: Steps in our Research Methodology

Table 2: Mapping between Firefox test status and categories used in this paper

Firefox Test Status	Normalized Categories	Number of test verdicts
PASS	*Pass*	120M
PASS(EXPECTED-RANDOM)	*Flaky*	265K
KNOWN-FAIL(EXPECTED-RANDOM)	*Flaky*	10K
FAIL	*Fail*	2M
UNEXPECTED-PASS	*Fail*	1K
UNEXPECTED-FAIL	*Fail*	705

3.4 Identifying HighFailureTests

The distribution of failures is not normal. Certain *HighFailureTests* account for a large proportion of total failures. Previous works have used this historical property to re-prioritize tests so that those that have failed frequently in the past will be run first [1, 10, 18]. We investigate if builds with *HighFailureTests* have an increase in the number of crash reports.

We define a *HighFailureTest* to be one that has failed on 10% or more test runs. As an example from the Firefox data set, a test "brokenUTF-16" ran 131K times and 79% of the total runs resulted with a "Pass" while 21% times it resulted as a "Fail". We consider this test to be a *HighFailureTests*. In contrast, the test "*hiddenpaging*" which ran 275K times passed 97% of the time with only 3% failures. This test would not be considered a *HighFailureTest* even though it has failed on past builds.

4 RESULTS

4.1 RQ1: Number of Crashes

How many crashes are there for builds on Dev, Beta, and Production?

The distributions in Figure 2 are the per-build number of crashes on development, beta and production channels. A box plot, is also contained within the distribution with the bottom and top of the box showing the 25th and 75th percentiles, respectively. The vertical line shows the median.

In total their are 2.8K builds associated with one or more crash reports and 3.8K builds that do not have any crash reports. Although the Dev channel contains experimental code and is likely not as stable as the other channels, we see fewer crashes on this channel. In the median case, development builds are associated with 0 crashes and with 3 crashes at the 75th percentile. The Beta channel builds are associated with a median of 437 crashes, and the Production builds are associated to 233 crashes.

Since builds on the Development channel are not typically run by main-stream end users, the number of total users is less likely explaining the limited number of crash reports for builds on this channel. As a result, we do not consider development channel in the remainder of this paper.

The purpose of the Beta channel is to stabilize code. Early adopters use these builds and provide crash and bug reports to help developers to stabilize the code. Despite having fewer users than the Production channel [15], builds on this channel have the highest number of crashes.

Code that reaches the Production channel has passed through various stabilization and bug fixing stages which are intended to reduce the number end user crashes. Although there are many crash reports, given the expanded number of users the production code does appear to be the most stable.

There are a median of 437 and 233 crashes for builds on the Beta and Production channels. Despite having more end users on the Production channel, there are fewer crashes likely indicating that production code has high stability.

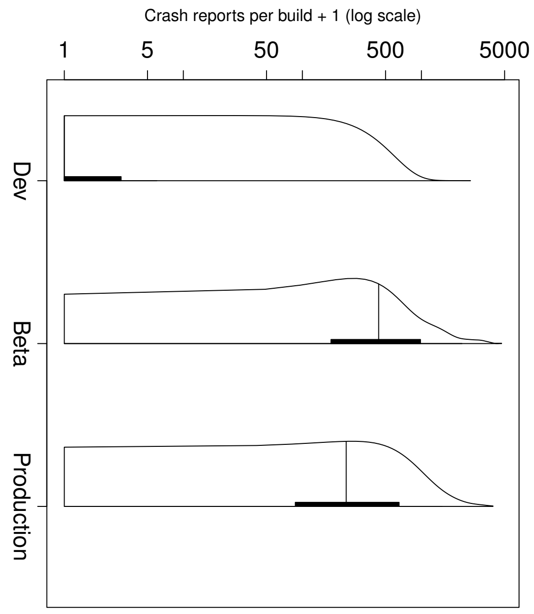

Figure 2: Number of crashes for each channel

4.2 RQ2: Test Failures

How many crashes are associated with builds that contain test failures?

Our conjecture is that when developer ignore quality assurance indicators there will be more crashes on these builds. In this research question, we examine the number of ignored test failures for builds and relate them with the number of crashes. We expect that more browser crashes will be associated with builds that have failing tests (*i.e.* failing builds) compared to those that do not have failing tests (*i.e.* passing or clean builds).

Firefox runs a large number of tests during each build. In the median case 3M tests are run on each build with a maximum of 11M. Table 2 shows the number of passing and failing tests: 120M and slightly over 2M, respectively. We exclude "random", non-deterministic tests examining them in the next section.

Figures 3 and 4 contrasts the number of crash reports for builds with at least one failing test with builds that have only passing tests. For the Beta channel, we found that builds that have failing tests have a median of 508 crash reports. In contrast, passing test builds have a median of 2 crash reports with 5 at the 75th percentile and a maximum of 25.

In the Production channel builds with failing tests have a median of 291 crashes. In contrast, passing build are associated with a median of 2 crash reports with 13 at the 75th percentile.

In the median case, builds that have failing tests are associated with 508 and 291 crash reports for the Beta and Production channels. In contrast, builds with passing tests have a median of only two crash reports, with some outlier pass production builds with many crashes.

4.3 RQ3: Flaky Tests

How many crashes are associated with builds that contain flaky tests?

Flaky tests fail in a non-deterministic manner and potentially hide bugs. For example, if a flaky test fails frequently, developers tend to ignore the failures and could miss the real bugs. We investigate whether ignoring flaky tests is a potential reason for increased browser crashes. We use the Firefox test outcome labels that contain "-RANDOM" to determine which tests are flaky (See the Methodology Section 3 for more details on the process of classification). In table 2, we see that 275K flaky test-runs are labelled with the "-RANDOM" verdict across all builds.

In the median case each Production channel build that contains at least one flaky test failure is associated with 234 crashes. There is a high degree of variation with 585 crashes at the 75th percentile and a maximum of 93k crash reports. Production builds with flaky tests are associated with almost the same number of crashes as those with regular failing tests.

For the Beta channel in Figure 3 we observed a similar patter with a 514, 1.2K, 21K crash reports for the median, 75th percentile and maximum, respectively. Beta builds with Flaky tests are associated with almost the same number of crashes as those with regular failing tests.

A Wilcoxon test comparing the crashes for flaky builds shows a statistically significant difference between the Beta and Production channels with the p-value $p < .001$. While future work is necessary, we conjecture that developers are more conservative with releasing production builds with known failing and flaky tests than with beta builds.

In the median case, builds with failing flaky tests we associated with 514 and 234 crash reports for Beta and Production, respectively.

4.4 RQ4: Historically HighFailureTests

Do failures of tests that have failed many times in the past lead to an increase in crashes?

In software systems, problems cluster around defective code and tests that have failed frequently in the past are likely to fail in the future [1, 10, 18]. To investigate these tests, we classify test that fail in 10% or more of their total runs as historically *HighFailureTests*. In the last distribution in Figures 3 and 4, we show that builds that have a failing test that is classified as *HighFailureTest* lead to lower quality builds as measured by an increase in reported crashes on both the Production and Beta channels.

The crash distribution for production builds that have one or more failing tests that are categorized as *HighFailureTests* show a medium, 75th percentile, and maximum of 780, 1.4k, and 21k crash reports, respectively. Production builds with *HighFailureTests* are associated with 2.7 times more crashes than builds with regular failing tests and 390 times higher than the passing builds.

For Beta builds the corresponding values are 585, 1.5k, and 92k crashes reports for the median, 75th percentile, and maximum, respectively. Beta channel builds with *HighFailureTests* are associated

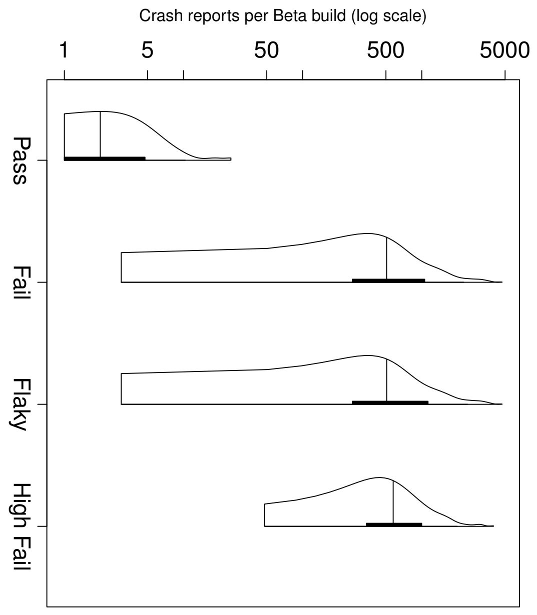

Figure 3: Crashes per type for Beta builds

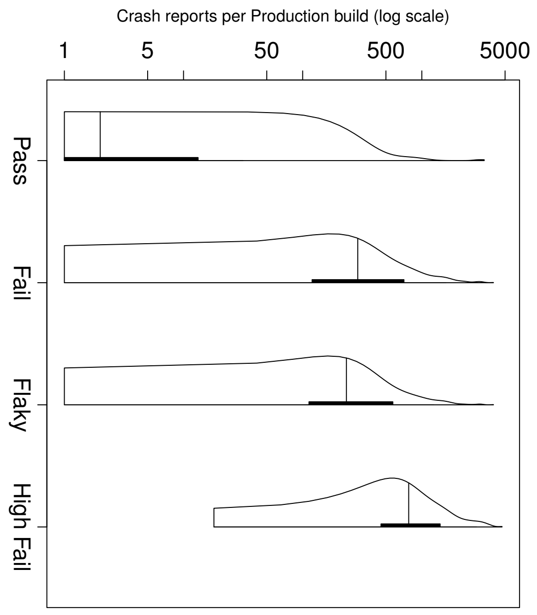

Figure 4: Crashes per type for Production builds

with 1.3 times more crashes than builds with regular failing tests and 292 times higher than the passing builds.

A Wilcoxon test comparing *HighFailureTests* on the Beta and Production channels shows a statistically significant difference between the crashes on these two channels with the p-value $p < 0.022$.

Unlike the flaky tests that are labeled by Firefox developers, *HighFailureTests* are not differentiated from other types of failing tests by the developers. This is especially problematic on production builds as these ignoring these *HighFailureTests* lead to many crash reports. Firefox developers might benefit from identifying and monitoring this classification of tests.

> *In the median case, builds that contain failing historically* HighFailureTests *are associated with 585 and 780 crashes for Beta and Production respectively.* HighFailureTests *are particularly problematic with production builds where there are 2.7 times more crash reports when compared to builds with normal test failures.*

5 RELATED WORK

We divide the related work into testing and build maintenance and quality. We are unaware of any work that has studied the impact of testing on field crash reports.

Testing and Flaky Tests. Labuschagne *et al.* studied the cost of regression testing in practice [11]. They found that 18% of the total test suite executions fail. More interestingly, 13% of these failures are flaky. Of the non-flaky failures, only 74% were caused by a bug in the system under test and the remaining 26% were due to incorrect or obsolete tests. They also found that in the failed builds, only 0.38% of the test case executions failed and 64% of failed builds containing more than one failed test. This study illustrates the importance of dealing with the flaky tests to improve the quality of the regression testing. Our study adds to this knowledge by studying the impact of test failures on field crashes.

Recent works on flaky tests identified the root cause of the flakiness. For example, Eloussi identified flaky tests from test results [2] in her doctoral research where she proposes three improvements for the basic technique to identify flakiness of tests. By manually examining the flaky test Eloussi divided the tests into three types: Non-Burstly, Burstly and State-Dependent Burstly. Another study on flaky test by Memon *et al.* [16] provides a detail post-mortem of flaky tests that provides actionable information about avoiding, detecting and fixing these types of non-deterministic tests. They inspect the test code by analyzing the code commits that likely fix flaky tests. They also identified the root causes of flakiness but not the impact after release. In our work, we measured the impact of flaky tests on crashes. Future work could use the crash reports on flaky tests to validate the causes identified by Eloussi and Memon.

General build system studies. Xin *et al.* performed an empirical study on bugs in build systems [17]. They categorized bugs based on their type and severities and found that the third highest percentage of bugs belong to the build-configuration category. They examined the association between the bugs and the build configurations, while we associate browser crashes with the failing tests in a build.

McIntosh empirically studied build systems in his dissertation [14]. His publications include a build maintenance study of the effort spent on maintaining the build process and the ownership of these build scripts [15]. He found that maintaining the build system required significant effort, with an overhead of 27% on source code development and 44% on test development. Our work adds evidence that build maintenance is an important problem by showing that ignoring build and test problems lead to substantially more crashes increasing developer effort and impact on end users.

6 CONCLUSION AND FUTURE WORK

In this paper, we investigate the association between builds and browser crashes on Beta and the Production channels of the Firefox web browser. We study the impact of ignoring failing, flaky, and *HighFailureTests* on the number of crashes for a build.

We observe that ignoring failing tests makes the Firefox builds much more crash prone compared to the builds that do not have any failing tests (passing builds). Passing builds have a median of 2 crashes for both Beta and Production. In contrast, builds with failing tests have 508 and 291 crashes in the median case, respectively. Flaky tests non-deterministically pass or fail reducing developer confidence in the test. In the median case, builds with failing flaky tests had 514 and 234 crashes for the Beta and Production channels, respectively. Previous works have shown that tests that have failed in the past are more likely to fail in the future [1, 10, 18]. We quantified *HighFailureTests* as those that have failed in 10% of past runs. In the median case, builds with failing *HighFailureTests* have 585 and 780 crashes for Beta and Production.

Our results show that ignoring failing and flaky tests results in more crashes in Beta than Production. However, ignoring *HighFailureTests* tests leads to more crashes on the Production than Beta channel. Ignored *HighFailureTests* were associated with a median of 780 crashes on the Production channel. This is the most crashes associated with any type of failing test and channel. In the median case there are 2.7 times more crashes per build than builds with normal test failures. A failing *HighFailureTests* clearly warrants a detailed investigation before release by Firefox developers.

We hope that our work will inspire developers to understand the high risk of ignoring failing tests. We hope that researchers will extend our work by examining both the root causes of failing and flaky tests and by contributing advanced statistical models to enhance our understanding of the associated risks.

ACKNOWLEDGEMENTS

We would like to thank Bram Adams from Ecole Polytechnique Montreal for providing the Firefox data and for helping us understand some its key features.

REFERENCES

[1] Sebastian Elbaum, Gregg Rothermel, and John Penix. 2014. Techniques for Improving Regression Testing in Continuous Integration Development Environments. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 235–245. https://doi.org/10.1145/2635868.2635910
[2] Lamyaa Eloussi. 2015. *Determining flaky tests from test failures*. Ph.D. Dissertation. University of Illinois at Urbana-Champaign.
[3] Mozilla Firefox. 2012. Test Labeling in Mozilla : integration-mozilla-inbound. http://bit.ly/2riO0Nh.
[4] Mozilla Firefox. 2012. Test Labeling in Mozilla : Xpcshell Self Test. http://bit.ly/2scySB1.
[5] Mozilla Firefox. 2017. Build:Release Automation. https://wiki.mozilla.org/Build:Release_Automation.
[6] Mozilla Firefox. 2017. Firefox Crash Reporter. https://support.mozilla.org/en-US/kb/mozillacrashreporter.
[7] Mozilla Firefox. 2017. Release Management/Release Process. https://mzl.la/1KhlZf9.
[8] Foutse Khomh, Bram Adams, Tejinder Dhaliwal, and Ying Zou. 2015. Understanding the impact of rapid releases on software quality. *Empirical Software Engineering* 20, 2 (2015), 336–373.
[9] Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. 2012. Do faster releases improve software quality?: an empirical case study of Mozilla Firefox. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*. IEEE Press, 179–188.
[10] Jung-Min Kim and Adam Porter. 2002. A history-based test prioritization technique for regression testing in resource constrained environments. In *Software Engineering, 2002. ICSE 2002. Proceedings of the 24rd International Conference on*. IEEE, 119–129.
[11] Adriaan Labuschagne, Laura Inozemtseva, and Reid Holmes. 2017. Measuring the Cost of Regression Testing in Practice: A Study of Java Projects Using Continuous Integration. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 821–830. https://doi.org/10.1145/3106237.3106288
[12] Eloussi Luo, Hariri. 2014. An empirical analysis of flaky tests. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 643–653.
[13] Mika V. Mäntylä, Foutse Khomh, Bram Adams, Emelie Engström, and Kai Petersen. 2013. On Rapid Releases and Software Testing. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM '13)*. IEEE Computer Society, Washington, DC, USA, 20–29. https://doi.org/10.1109/ICSM.2013.13
[14] Shane McIntosh. 2015. *Studying the Software Development Overhead of Build Systems*. PhD dissertation. Queen's University.
[15] S. McIntosh, B. Adams, T. H. D. Nguyen, Y. Kamei, and A. E. Hassan. 2011. An empirical study of build maintenance effort. In *2011 33rd International Conference on Software Engineering (ICSE)*. 141–150. https://doi.org/10.1145/1985793.1985813
[16] Cohen Memon. 2013. Automated testing of gui applications: models, tools, and controlling flakiness. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 1479–1480.
[17] X. Xia, X. Zhou, D. Lo, and X. Zhao. 2013. An Empirical Study of Bugs in Software Build Systems. In *2013 13th International Conference on Quality Software*. 200–203. https://doi.org/10.1109/QSIC.2013.60
[18] Y. Zhu, E. Shihab, and Rigby PC. 2018. Test Re-prioritization in Continuous Testing Environments. In *2018 IEEE International Conference on Software Maintenance and Evolution*. 10.