2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)

A Comparison of Bugs across the iOS and Android Platforms of Two Open Source Cross Platform Browser Apps

Wajdi Aljedaani*, Meiyappan Nagappan†, Bram Adams‡and Michael Godfrey†
*Dept. of Computer Technology, Al-Kharj College of Technology, Al-Kharj, Saudi Arabia
Email: waljedaani@tvtc.gov.sa
†Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
Email: {mei.nagappan, migod}@uwaterloo.ca
‡Dept. of Computer and Software Engineering, Polytechnique Montréal, Montréal, Quebec, Canada
Email: bram.adams@polymtl.ca

***Abstract*—Mobile app developers want to maximize their revenue and hence want to reach as large an audience as possible. In order to do this, they need to build apps for multiple platforms - like Google's Android and Apple's iOS, and maintain them in parallel. Past research has examined properties of the issues addressed in either Android or iOS, but not to compare the work between both. Our main motivation has been to determine if there were differences in how issues manifest themselves in iOS and Android, when we control for the projects, by considering the same apps across multiple platforms. In this paper, we compare issues across two mobile platforms — iOS and Android — for two open source browsers — Mozilla Firefox and Google Chromium. We consider three dimensions of study: frequency of issue report submission, fixing time of issues, and type of issues (using topic modeling on the issue description to generate the categories). We found that there were indeed differences; in particular, we found that there were more issues in the Android version of the apps and the gap with the iOS version is increasing. We observe that in both apps the fix time and type of issues are different for each platform. We also noted certain kinds of issues that may be more prevalent for different browser/platform combinations. This can advise project leads in identifying and allocating development resources to address key problem areas. Hence, issue reports seem more dependent on the platform than on the mobile app, making development and maintenance effort hard to estimate.**

***Index Terms*—Issue repository, issue reports; Mozilla Firefox; issue fixing; Google Chromium; empirical studies; topic model.**

I. Introduction

In the ten years since their introduction, mobile devices such as smart phones and tablets have become highly popular and powerful social technologies. Currently, two deployment platforms dominate the mobile market: Apple's iOS, a (mostly) closed-source operating system, and Google's Android OS, a (mostly) open-source operating system. Unsurprisingly, many mobile developers build applications (apps) that target both platforms. Researchers have compared the development of such applications [21], but to date there has been no study of how issues for such systems manifest themselves across the two deployment platforms. Indeed, most prior studies on issue repositories have targeted desktop and server applications [18], as well as apps written for the Android platform [5]. To the best of our knowledge, this is the first empirical study that analyzes issue reports, issue-fixing time, and issue types for applications that target both the Android and iOS platforms.

In this work, we perform a cross-platform investigation into the frequency, fix time, and types of issues associated to two browsers, Chromium and Firefox. Our motivation is pragmatically aimed: such knowledge could aid a mobile app developer to identify and allocate resources between the platforms, and support better informed management of the software products. Also, such research could help to identify the types and rates of issues across the different platforms, for example whether Firefox on iOS was found to have more usability issues than Firefox on Android.

Chromium and Firefox were natural choices for this study because they are popular and both offer rich, detailed, and freely available issue repositories as datasets, and are from the same domain. These repositories provide a wide array of supporting data on the types and eventual solutions for the issues identified by the end users and application developers; comprehensive cross-platform issue datasets such as these are hard to come by.

Specifically, we investigated three research questions in our study:

RQ1: *How are the numbers of reported issues distributed across the deployment platforms?*

We found that both browser apps written for the Android platform had more issues reported than their iOS counterparts by a significant margin, with more issues for Firefox compared to Chromium. However, Chromium-Android in 2015 and 2016 had considerable increase in the quarterly rates of issues reported, while Firefox-Android has held a more consistent pattern. In contrast, while the iOS apps have fewer reported issues than the Android apps, the trends for the browsers are similar: in Firefox, iOS has more issues, but the numbers have decreased over the years, while Chromium-iOS has fewer issues but are increasing.

978-1-7281-3395-9/19/$31.00 ©2019 IEEE
DOI 10.1109/MOBILESoft.2019.00021

RQ2: *How are issue-fix times distributed across the deployment platforms?*

We found that in Firefox, Android issues took longer to fix than iOS issues. However, in Chromium, Android issues took less time to fix than iOS issues. The longest wait periods before being fixed are in Firefox, while Chromium issues tend to be finished more quickly. Firefox-iOS issues took less time to fix compared to Chromium-iOS.

RQ3: *How are issue types distributed across the deployment platforms?*

We performed topic modelling on the textual issue descriptions to infer the underlying types of the issues; we examined the generated topics and chose labels that seemed to fit best. Using these labels, we found that Firefox-Android has more issues related to *Multimedia* and *Bookmarks*, while Firefox-iOS has more issues related to *Bookmarks* and *Post-Release Failure*. In Chromium-Android, more issues are related to *Device-Specific Failure* and *Crashes*, while *Pre-Release Failures* and *Third Party Library* issues occur more on the Chromium-iOS platform. Hence, there is not a consistent type of issue that is prevalent to a platform across both case study systems. However, developers can still use such a topic analysis to find what types of issues occur frequently in their app, and recruit the required expertise.

The remainder of the paper is organized as follows. Section II presents background information on issue-tracking systems and an overview of Firefox' and Chromium's development and release processes on Android and iOS. Section III illustrates our case study approach, data collection, and processing. Section IV presents and discusses the results of our three research questions. Section V discusses related work, and we describe threats to validity in Section VI. Finally, we summarize our findings in Section VII.

II. Background

Here we discuss background information about issue-tracking systems as well as about the two subject systems of this study.

A. Issue-Tracking Systems

Issue-tracking systems — also called bug-tracking systems — enable the management, tracking, and resolution of programming issues in large-scale software projects. End-users submit their issue description in a report, often auto-generated by the software in use, for developers to examine and possibly patch. These reports are collected, examined, triaged; if the triager decides that the report describes a problem that needs attention, the report is assigned to a member of the development team for fixing. Once addressed, the reports are archived such that they can be consulted if-and-when they become relevant again in the context of a future issue report.

The stages in the lifecycle of an issue report are fairly simple: first, the user submits a report of the symptoms and replication steps of an issue via an online form, which assigns the report for *triage* (review). Apart from the textual description of an issue, an issue report typically also records relevant metadata, such as dates, the reporter's name, and the OS version used when the issue occurred. Triagers will then examine and evaluate the issue report to determine if it represents an issue worthy of attention; to do so, triagers exploit their contextual knowledge of the system, the developers, and the project's history. The outcome could be that the issue is deemed relevant and worth attending to, *irrelevant* ("WONTFIX"), *non-reproducible* ("WORSKFORME"), or a *duplicate* of a known issue that has been fixed or is actively being worked on. If it falls under relevant but unattended to, the triager will assign a developer to start addressing the issue report. The developer can interact with the issue reporter and any interested stakeholder through issue report comments, IM, email, or other means provided by the issue tracking system that managers the reports. Once finished, the report can be *closed* if the issue is considered to have been fixed, *abandoned* if a fix is impossible or impractical, or *reopened* when an accepted fix is found to be inadequate.

B. Overview of Firefox and Chromium

In this case study, we focus on two of the larger open-source issue report repositories that span multiple mobile platforms. We chose the domain of web browsers, since these applications are supported on multiple platforms and system environments. In particular, we selected Mozilla Firefox and Google Chromium, as they are open source projects in wide use whose development practices can be tracked and analyzed by anyone. In the remainder of this section, we now discuss how the Firefox and Chromium projects are being developed, and how these processes differ between their Android and iOS versions

Software Goals. Mozilla Firefox and Google Chromium are widely used web browsers. The first official version of Mozilla Firefox was released in 2004, although much of that codebase originated in another open source project, called Mozilla[1], started by Netscape in 1998. One notable goal of Firefox is to be compliant with various web standards, as well as having extensive features and a modular design [28]. Firefox is developed and released under the Mozilla Public License (MPL).

The first release of the Chrome browser was a desktop version in 2008, followed by the first mobile version in 2011. Chromium is built upon the Webkit framework, like many other modern browsers including Safari but (notably) not Firefox. It should be noted that there are two variants of the Google browser: *Chrome* and *Chromium*. Chrome is the Google-branded variant that is used by the vast majority of end-users; it is a closed source project, but is based on its open source cousin Chromium, which is developed and released under the BSD license. Google's Chrome extends Chromium with a number of proprietary features, including support for multiple media formats.

[1]The original Mozilla project combined several applications in one unified system, including a web browser and an email client. The project architects decided to separate these out into individual, independent applications called Mozilla Firefox, Mozilla Thunderbird, etc.

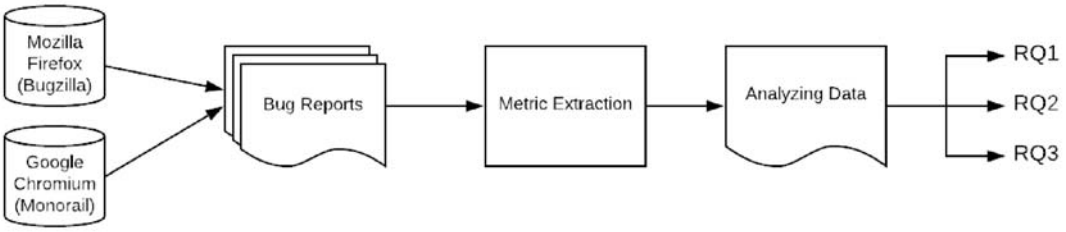

Fig. 1: Overview of our approach to extract the metrics of issue reports and analyze the data.

Chromium has other advantages, such as not collecting and forwarding usage data and the ability to turn off the security sandbox on Linux platforms. We have studied Chromium in this project as a proxy for Chrome, since we cannot access data from the closed source parts of Chrome. However, as the vast majority of the key functionality of Chrome is inherited from Chromium, we feel this is a reasonable choice.

While the main feature sets of Firefox and Chromium are comparable, users may perceive differences between them in terms of battery usage, start-up time, privacy/security (open source project vs. Google-backed project [29]), and availability of plugins. Firefox and Chromium both provide a mobile version of their app for both the Android and iOS platforms. Since Android version 4.4 ("KitKat") released in 2013, Chrome has been the default browser for the Android platform; Firefox must typically be downloaded and installed separately if desired by the user. For iOS, the *Safari* browser is the default. While many iOS users are content to use that app, some users also download and install Chrome and/or Firefox instead. We have not studied Safari as it is (mostly) a closed source project and it has not been ported to Android.

Release Frequency. Table I shows the summary of the release cycle times of all 4 apps [22]–[25], i.e., the number of days between successive official releases. These numbers do not distinguish between major and minor versions, but include only the official releases customers could download from the official app stores, excluding any non-official release such as beta releases. We can see how the median release cycle times of Firefox for Android and Chromium for iOS are similar (about 3 weeks median), yet much larger than the median cycle time for their Android counterparts (about 2 weeks median). Again, these differences could indicate differences in the way issues are being addressed across platforms, since projects with shorter cycle time might have less time for quality assurance [32].

III. STUDY DESIGN

This section describes the approach of our study, as well as how we collected and processed the data to address our three research questions.

A. Study approach

This section presents the design of our study on the issue repositories of the iOS and Android versions of Firefox and Chromium. Figure 1 shows an overview of our approach. For each case study system, we first extract the necessary data from the entire issue repository. Then, we identify the issue reports related to a specific platform, and calculate relevant metrics for those issue reports. We next analyzed the issue report metrics to identify, for each app, possible differences between the reports of its Android and iOS versions. The specific metrics studied in this paper are described in detail in the approach subsections of each research questions (see Section IV).

B. Data Collection

We collected the data of all issue reports of Firefox submitted for its Android and iOS apps in the Mozilla project's Bugzilla issue repository[2] for the period from January 1, 2014 to July 23, 2016; this yielded a total of 14,082 issues. The details of our datasets are listed in Table II. In addition, we gathered the data of all the issue reports of Chromium that were submitted to its Monorail issue tracker [3] for the period from March 3, 2015 to July 24, 2016; this yielded a total of 8,484 issues.

C. Data Processing

First, for each issue repository, we extracted a set of data: *issue ID*, *status*, *summary*, *OS*, *reported date*, and *closed date*. We used the R programming language to perform the corresponding analyses in the three research questions. Although Firefox has recorded its issues since 2002, the Chromium project started more recently, in 2015. To make more reliable and accurate comparisons between both software systems, we filtered the Firefox issue report data to include only issues reported between January 1, 2014, and July 24, 2016.

IV. STUDY RESULTS

This section presents and discusses the results of our analysis. For each research question, we present the question, the analysis approach, the results of the analysis, and we discuss the findings.

[2]https://Bugzilla.mozilla.org/
[3]https://issues.chromium.org/hosting/

Project	Platform	Min.	Q1	Median	Mean	Q3	Max.
Firefox	Android	1.00	6.25	13.50	17.41	26.00	49.00
	iOS	4.00	10.25	24.00	31.05	43.00	131.00
Chrom.	Android	1.0	5.0	11.5	17.7	25.0	79.0
	iOS	1.00	14.00	21.00	25.56	34.50	74.00

TABLE I: Summary of the time between successive releases (in #days) for the four apps.

System	Platform	#Issue Reports	#Common Overlap Issue Reports	Start Date	End Date
Firefox	Android	10,852	0	01-01-2014	23-07-2016
	iOS	3,230	0	05-12-2014	23-07-2016
Chromium	Android	7,781	42	03-03-2015	24-07-2016
	iOS	703	38	03-03-2015	24-07-2016
Total	**22,566**				

TABLE II: Statistics of the data sets.

RQ1: *How are the numbers of reported issues distributed across the deployment platforms?*

Motivation. In this research question we want to know how many issues are reported for each platform. By knowing this we could understand if the same app being built for different platforms need the same or different amount of resources. If the number of issues are different consistently, then irrespective of why the number of issues are more, we can conclude that more resources are needed.

Approach. Collecting the dataset for the Mozilla Firefox-related issues was simple, due to the intuitive structure of the Mozillan issue report system; all Firefox issues tagged with *Android* and *iOS* were searched for and downloaded, resulting in a preliminary database of 102,063 issue reports. Although the Mozillan issue repository archives date back to 2002, we decided to focus on a narrower, more recent period of January 1, 2014 to July 24, 2016; these particular dates mirror the period for which we could also extract issue reports for Chromium, a much newer project. The Mozilla dataset for this date range consisted of 14,082 issue reports, i.e., 10,852 for Android and 3,230 for iOS (cf. Table II).

Collecting the equivalent dataset from the Chromium issue repository was more difficult. As the Chromium issue repository does not allow for the same degree of nuance in the search function (i.e., it did not necessarily allow straightforward searching for specific categories of issues), it allowed only for the selection of a number of issue reports in a particular time range. To work around this limitation, the initial Chromium-related dataset was culled from the Google Chromium database by selecting all issue reports from the time period of interest. Once this selection was complete, the dataset comprised 50,078 reports. The dataset was then processed in order to allow for cross-platform comparison between the two dataset sources, as follows.

First, Chromium issue reports that contained multiple OS names and related terms in the OS descriptor field were duplicated, with one for each relevant OS. For example, if an issue report contained the names of *Android*, and *iOS* in the *OS* field, then an additional copy would be made of the report, and a copy of each would be entered into the appropriate OS database. This step enabled analysis of issue frequency across the different OS types; it resulted in 5,923 "new" reports being added, making a total of 56,001 reports in the dataset altogether.

Step two in the processing stage was to remove all issue reports that did not contain a specific OS name or a related term in the relevant field of the issue report. These reports were discarded, as being irrelevant to the focus of this research. This included issues with the "*[empty]*" descriptor in the *OS* field. This stage eliminated 13,000 reports from the dataset, bringing the total number of issue reports down from 56,001 to 43,001. Once this stage was complete, the issue reports for Android and iOS were retained, while the other OS-related issue reports were discarded. This collection was then used as the basis of the analytical dataset for Chromium, resulting in a total of 7,781 issue reports for Chromium-Android and 703 reports for Chromium-iOS.

Results. Figure 2 shows the number of issues in Firefox for both Android and iOS versions. During the first three quarters (Q1–Q3) of 2014, there were a substantial number of issues reported on the Android platform. However, as seen in Table II, no issue data exists for the iOS platform for that same period; this is because the iOS version of Firefox was not officially released until Q1 of 2015. During Q4 of 2014, the first (small number of) issues for the iOS version were reported.

Throughout the four quarters of 2015, the number of issues reported for Android and iOS is more or less similar, with small fluctuations. Only in Q4 of that year the number of iOS reports dropped substantially. While this number remained constant in Q1 and Q2 of the next year, the number of reports for Android saw a large increase. Note that the number of issues for Q3 of 2016 is small due to the cutoff date of our analysis.

For Chromium (Figure 3), we started collecting data from Q1 of 2015. Overall, the number of issues reported for iOS is substantially lower than for Android. In Q1, the number of issues reported for Android is substantially lower than in later quarters. In Q2 of 2015, there was a large increase for the Android devices from 430 to 1,350 compared to a much smaller proportional increase in number of issues in the iOS platform from 41 to 66. In Q2–Q4 of 2015 and Q1–Q2 of 2016, the number of issues increases over time in Android

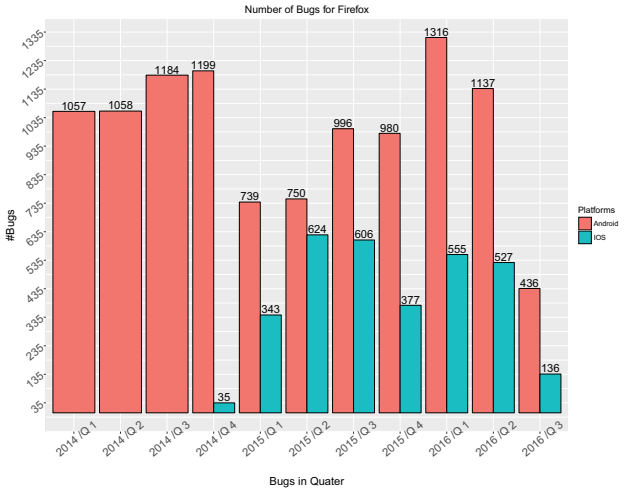

Fig. 2: Distribution of the number of issues for Firefox.

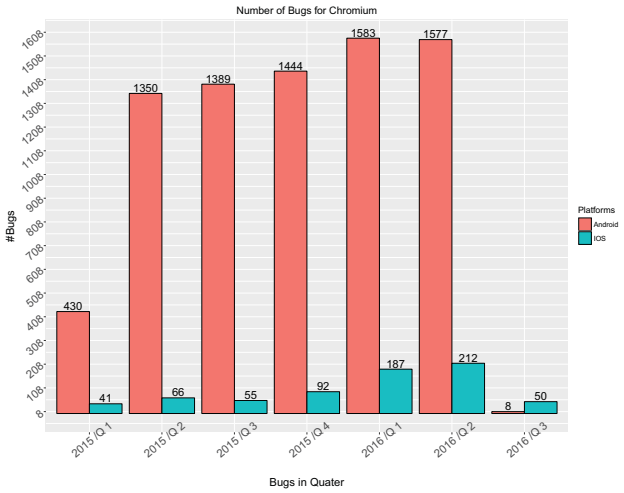

Fig. 3: Distribution of the number of issues for Chromium.

and iOS, except for a brief decreases in Q3 of 2015 from 66 to 55. In Q3 of 2016, the number of issues for both Android and iOS drops due to the cut-off date of our data set.

Discussion. By mining the repositories of Chromium for patch authors, we find that the two Chromium apps share 57 developers (Table III), on top of which there are 450 Android-only and 60 iOS-only developers. On the other hand, the two Firefox apps only share 15 developers. These are assisted by 251 Android-only developers and 56 iOS-only developers. These differences in development make the Firefox and Chromium apps interesting to compare in this study, since they might have repercussions in the way issues are being addressed.

The trends that we observed in Figure 3 confirm the large overlap between the iOS and Android teams of Chromium that we saw in Section II, as the number of issues evolved

Project	Platform	#Common Team Members	#Unique Team Members
Firefox	Android	15	251
	iOS	15	56
Chromium	Android	57	450
	iOS	57	60

TABLE III: Team statistics in Firefox and Chromium for both the Android and iOS apps.

proportionally, albeit at different levels of magnitude The reason why the number of issues is the lowest in Q3 of 2015 is the introduction of new features in Chromium-Android related to tabs, security, and plugins from Q4 2015 to Q2 2016.

One further point that should be considered in this perspective, however, is that the total number of users for Android is significantly higher than that of iOS, due to the larger range of devices supporting Android, as well as Android's open source nature and that iOS uses the Safari browser as its default. As shown by Herraiz et al. [33], this can explain a large part of the different orders of magnitudes in number of reported issues

In summary, there are consistently more issues for the Android platform in both Firefox and Chromium . There are likely many contributing factors for this: the Android platform has more users than iOS, and many iOS users do not bother to install Firefox or Chromium on their devices, so fewer issues may be noticed and reported; and the Android platform supports a significantly larger and more varied set of devices than iOS, introducing a lot of complexity that apps must handle in their code base, which is not true for iOS apps. For these reasons, software managers may wish to closely monitor the amount of development resources allocated to each platform, to determine where they may be most effectively deployed.

RQ2: *How are issue-fix times distributed across the deployment platforms?*

Motivation. In this research question we want to examine how long it takes to fix an issue in a given platform. We want to explore this to see if the types of bugs or the priority to fix them are the same across platforms. We do not know why the bugs might take longer to fix, but what we may know is that different platform's issues may take different amounts of time to fix.

Approach. The data was sorted and analyzed to determine the time taken to repair a given issue, which we defined to be the number of days between being first opened and being marked as "Resolved" (Firefox) or "Fixed" (Chromium). Issues that were not marked with either of the two above statuses were excluded from consideration, as the focus of our study is on issues that have been repaired. Once this selection was complete, the size of the dataset that was examined for analyzing the fixing time (i.e., the number of issues analyzed) for Firefox-Android is 2,670 and for Firefox-iOS 1,700, while for Chromium-Android it is 1,775 and Chromium-iOS 218. We then used a Wilcoxon pairwise analysis on the data subsets to determine which browser/platform combination possesses the longest and shortest times to repair.

Company	Platform	Min.	Q1	Median	Mean	Q3	Max.
Firefox	Android	0.00	3.25	12.00	42.95	37.00	800.00
	iOS	0.00	2.00	8.00	31.02	33.00	390.00
Chromium	Android	0.00	4.00	21.00	96.99	100.00	1070.00
	iOS	0.00	13.0	81.5	206.1	372.2	1008.0

TABLE IV: Summary of fixing time of the projects (in #days).

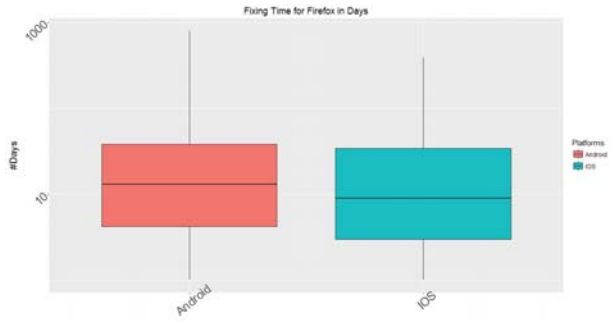

Fig. 4: Box-plot of fixing-time (in #days) of issue reports for the iOS and Android platforms (Firefox).

Results. Figure 4 shows a box-plot of the Firefox issue resolution time. The median number of days taken to fix issues was higher for Android compared to iOS. On average it takes a median of 12 days to fix an issue on Android compared to 8 for iOS (see Table IV). Out of 2,670 issues in total almost 269 were solved in less than a day or so for the Android version.

Figure 5 shows the corresponding box-plot for Chromium issue resolution in number of days. Here we can see that the median number of days used to fix issues was substantially higher in iOS compared to Android. The median time to resolve issues for Chromium-Android is around 21 days, with almost 389 out of 1,775 issues being resolved in less than a day.

Discussion. We performed a Wilcoxon Rank-Sum hypothesis test to determine whether the observed differences in the distribution of issue fixing time are significant. From the results in Table V, we can see that in Firefox, Android issues took a significantly longer time to fix than iOS issues. However, in Chromium, Android issues took significantly less time to fix than iOS issues. Given that the Android apps both have a similar much shorter release cycle time than their iOS apps, it is surprising that Firefox' median issue fixing time on iOS is lower than the time for Android.

One possible explanation could be related to the findings of Khomh et al. [32] for the desktop version of Firefox related to the time required to fix bugs before and after the switch to more rapid releases. They observed that bugs were fixed faster, but that proportionally less bugs were fixed. Bugs that could not be addressed for the upcoming release, would be pushed towards the next one. Given the much smaller development team working on the Android version of Firefox than on the Android version of Chromium (including both the common and unique team members), we conjecture that more issues are being propagated to later releases, effectively

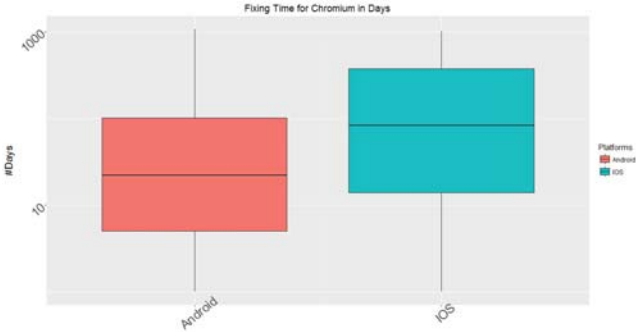

Fig. 5: Box-plot of fixing-time (in #days) of issue reports for iOS and Android Platforms (Chromium).

Company	X vs. Y	Less	Greater	2-sided
Firefox	Android vs. iOS	1	1.852e-11	3.704e-11
Chromium	Android vs. iOS	$<$ 2.2e-16	1	$<$ 2.2e-16

TABLE V: P-values for Wilcoxon Rank-Sum tests.

increasing the fix time. In any case, on the basis of our study, we cannot conclusively say whether Android issues take more or less time to fix as compared to iOS issues, i.e., both platforms can be associated with longer issue fixing times.

Despite this observation, one characteristic that all of the tested OS/Browser combinations have in common is that they all are heavily left-skewed with an extended right tail. In other words, the vast majority of issues are simple repairs that can be taken care of in a very short period of time; this means that the gradually descending curve may be more representative of developer bottleneck than of actual time taken to repair the issue. The results suggest that there may be room for improvement in the various issue-repair processes (e.g., triaging, assigning, etc.); while it seems that some issues are truly difficult to fix, they are a minority in comparison with the fairly basic issues that consume the majority of the developers' attention resources.

RQ3: *How are issue types distributed across the deployment platforms?*

Motivation. In this research question we want to examine if the types of issues raised in each platform are similar or different. By knowing this, we may be able to decide if similar or different expertise is required from app developers of different platforms.

Approach. The issue report descriptions were processed for relevant keywords, using the latent Dirichlet allocation (LDA) method. By considering the set of issue report descriptions as a corpus of documents, LDA identifies clusters of words that co-occur often enough across documents (issue reports) as "topics". Each issue report can then be interpreted in terms of the topics it contains, similar to how newspaper articles can discuss topics such as sports and finance.

We used Gibbs sampling with default parameters, except for two: the "thinning" parameter and the number of topics [20].

Gibbs sampling [27] is based on random search, hence it is normal to discard the first few results as they are unlikely to be a true reflection of the corpus being analyzed; this is called the *burn-in period*. We used a burn-in period of 4,000 iterations, and executed the sampling 5 times, with different starting points for each. To avoid correlation between samples, we set the thinning parameter to 500. This means that the Gibbs sampling selects every 500^{th} iteration (of the 2,000). We also experimented with the number of topics by comparing the results with different selected features. This helped us in inferring the best topics from the corpus.

In particular, the LDA analysis calculates for each issue report a vector containing for each topic the probability that it is "covered" by the issue report. We chose 10 as the number of topics in our study. The LDA analysis also determines the terms that are most relevant to a given topic. We assign each issue report to its most important topic (highest probability). Then, we collect the number of issues in each issue topic and compare the number of issues across the iOS and Android platforms for each case study system.

We randomly selected 3,000 issues from Chromium (2,297 bug reports sampled from the Android platform, and 703 bug reports from iOS) and 3,000 from Firefox (1,500 for each platform) to perform the LDA analysis, since this represents a sample with a 99% confidence level and a 2% confidence interval [4]. We did not run the LDA analysis on all the bugs despite the approach be fully automatic because, there were more bugs in Firefox than in Chromium. Hence, we thought that the difference in frequency would bias the results towards Firefox topics. We then combine the 6,000 issues and run a combined LDA analysis. We do a combined LDA analysis so that we can compare the topics across the two platforms for both the apps. In the first run, we found that the keywords, *Android*, *iOS*, *Firefox*, and *Chromium* were part of the topic keywords. Since we did not want these keywords to influence the LDA analysis (the goal of which is to extract the types of issues in each platform for each of the two case study systems), we removed them and re-ran the LDA analysis.

Results. Table VI shows the 10 topics and the keywords that LDA selected as most representative of the topics. Figures 6 and 7 show the percentage of issues that are present in each platform for each case study system. Below, we will discuss the topics with the largest differences between the two platforms of Firefox and Chromium.

The largest differences between Android and iOS platform are shown for Chromium in Figure 7 for Topics 7 and 9. These topics show similar, but smaller, trends for Firefox in Figure 6. Referring to Table VI, we can see that Topic 7 concerns content that fails to work on some devices, while Topic 9 (more prevalent on iOS) appears to concern offline use. The fact that Android has to deal with a heterogeneous set of devices explains why Topic 7 is more prevalent on Android, while it is unclear why offline use is more of an issue for iOS apps.

[4]https://www.surveysystem.com/sscalc.htm

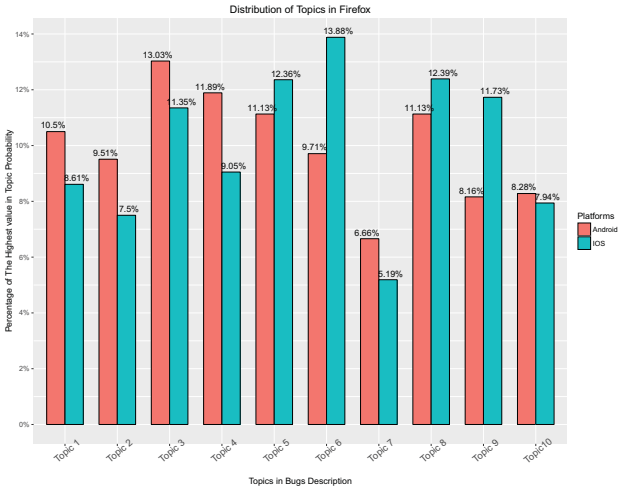

Fig. 6: Distribution of the number of issues in each topic in Firefox.

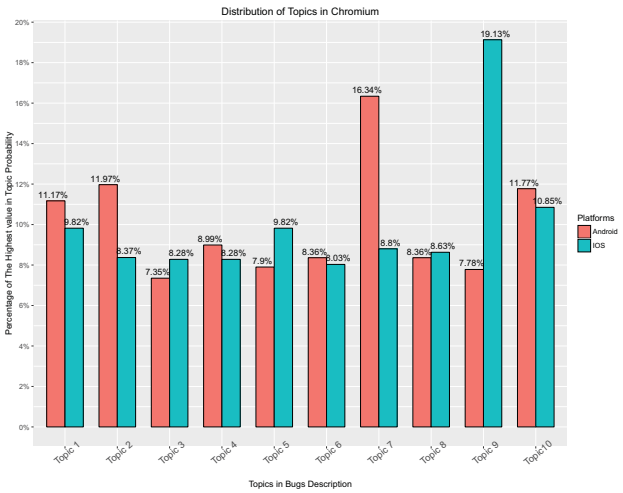

Fig. 7: Distribution of the number of issues in each topic in Chromium.

Smaller, but still substantial differences between Android and iOS are shared between Firefox and Chromium for Topics 2 and 5. Topic 2 appears to concern playing and controlling video content, while Topic 5 appears to concern browser tabs and how many can be opened. Again, it is unclear why Android is more prone to video content issues (Topic 2) than iOS, and iOS more prone to tab issues than Android. In the worst case, these observations could help practitioners to plan for issues related to these functionalities.

Finally, we also observe a large difference in percentage of issues for Topic 6 in Firefox, while no such difference exists for Chromium. Topic 6 concerns the menu bar, and we can see that in Firefox, iOS has considerably more issues related

	Topic1	Topic2	Topic3	Topic4	Topic5	Topic6	Topic7	Topic8	Topic9	Topic10
1	page	video	update	add	tab	menu	content	url	offline	update
2	show	doesnt	bookmark	sync	open	view	fail	search	build	crash
3	load	work	site	data	link	list	test	bar	load	app
4	web	play	panel	disable	support	mode	device	scroll	change	webview
5	content	button	screen	allow	close	implement	run	text	local	file
6	dont	control	icon	client	display	history	failure	select	code	error

TABLE VI: Summary of the terms in 10 topics from the LDA analysis of Firefox and Chromium bugs in the iOS and Android platforms.

to this topic, while in Chromium, Android has marginally more issues. Without more detailed — and largely manual — analysis, it is hard to draw further conclusions from these results. The remaining 5 topics of Table VI do not have considerable differences in percentages of issues.

Discussion. In conclusion, there are some types of issues that appear to be prevalent in both the iOS or Android platforms; this information could be used by the respective development teams to budget more time and/or resources for features or maintenance related to prevalent issues. Additionally, we have demonstrated that app developers could potentially use topic analysis approach as we have done to determine if a specific type of issue is more likely to be present in one platform over the other.

V. Related Work

Issue repositories are a vital source of development knowledge and for large, long-lived software systems. They permit users to inform developers of the issues that the users experienced while utilizing the software. There are many studies that relate to issue repositories; most of the papers are targeted to relating the level of communication amongst committers to issue-proneness [1], evaluation of issue report quality and content [2], and developer prioritization to rank the contribution of repository developers [3]. In this section, we present related work concentrated on different aspects of issue repositories. We discuss three areas of related work and overlap, as well as present the differences between the related work and our work.

A. Mobile Issues Studies

Maji et al. [4] analyzed the reported cases of failures of two operating systems, Android and Symbian, based on issue reports. These reports had been provided by end users, third-party developers, and documentation of issue fixes from Android developers. The researchers analyzed 233 issue-fixes from 29 projects in the Android OS repository from October 2008 to October 2009. Their approach was to categorize the issues into several types of code modifications which required fixes in the source code. Their results showed that 23% of the issues needed significant source code changes, while 77% of the issues required minor modifications.

In a study similar to ours, Bhattacharya et al. [5] performed an empirical study on issues on the Android smartphone platform and 24 open-source Android apps from various categories such as health-fitness, tools, and communications. They applied their research on issue reports, issue fixes, and security issues. Their approach for measuring the issue-fixing time was structured as follows: they subtracted the time taken from the instant where the issue was reported and to the moment where the issue closed to get the actual length of time it took to resolve the issue in months. They found that, for the majority of the apps, the average time taken to fix issues ranges from 0 to 1.5 months.

Our approach is similar to theirs, but instead of concentrating only on issues on the Android platform, we also examine and contrast issues on the iOS mobile platform. Furthermore, we performed our case study on both Firefox Mozilla's and Google Chromium's issue reports. Thus, our dataset is larger and more varied than the one contained in prior studies.

B. Empirical issue studies and Qualitative Analysis

Zhou et al. [26] analyzed the similarities and differences in issues and the processes in fixing the issues between desktop and smartphone platforms. The study was applied on 88 open source projects on desktop, Android, and iOS. Their study was performed on issue report features, while in our study we studied all the issues in the repositories, including both issues and features.

Banerjee et al. [6] compared two large open-source issue repositories, i.e., Eclipse and Mozilla, and identified similarities and differences between them. The aim of their study was to analyze the type of reports, user behavior, repository structure, duplicate groups, and the frequency of report submission. Our approach is similar to theirs in the frequency of report submission; while they examined Eclipse and Mozilla, we concentrated on Mozilla and Chromium. We examined the evolution of issue repositories over time by analyzing the number of submitted report per year.

Various Studies on the qualitative analysis of issue repositories using a survey, questionnaires, and open-source projects [7]–[12] have been done in the past. An et al. [13] examined the characteristics of highly impactful issues in Mozilla Firefox and Fennec Android. They proposed statistical models to assist the software organizations to predict the highly impactful issues early before affecting a large number of users. Furthermore, they compared the issue fix rate of highly-impactful issues vs. other issues.
Their results found that the prediction models can accomplish a recall up to 98.3% in Android and precision up to 64.2% in Firefox.

Zaman et al. [41] studied issues in the desktop versions of Mozilla Firefox and Google Chromium to observe the collaboration among project members in terms of detecting and fixing performance issues. Their approach performed a comparison on a random sample of 400 performance and non-performance issue reports across four dimensions (Impact, Context, Fix, and Fix validation) and 19 sub-dimensions. Their result showed that there should be better support for collaborative cause analysis process, as well as more analysis of the impact of changes in performance.

Hu et al. [34] Studied 68 hybrid app-pairs in both Google Play store and Apple App store. They investigated the examined apps to identify whether tools for hybrid development achieve their main purpose. Hu et al. [35] Investigated if cross-platform apps accomplished a consistent level of star ratings and user review. They examined one and two star reviews which count as 9,902 reviews. Ali et al. [36] Presented a large-scale study of a cross platform mobile applications. They investigated 160K android and iOS applications. Their aim is to compare their app-store attributes, for example, stars, versions and prices.

Our work is different from their work in both purpose and approach. Our purpose is to compare issue reports across two mobile platforms instead of only between two projects. Moreover, we focused on the analysis of issue fixing-time to identify if these issue repositories can help Firefox and Chromium to improve the issue-fixing process.

C. Topic Modeling

Topic models have been used widely in software engineering research. Prior studies have used topic model for source code [14], issue localization [15], and duplicate issue detection [16], [17]. Khalid et al. [37] Studied the complaints of iOS users of 20 free iOS applications. They examined the low rating one and two stars reviews that counts as 6,390 user reviews. Their results found that 11% of reviews are complaints about recent update of the app. Martinez [38] Presented two dataset of questions and answers related to the development of mobile applications using Xamarin. Rosen et al. [39] Studied the data from Stack Overflow to examine what mobile developers ask about. They found that developers asked about distribution, mobile APIs, sensor and mobile tools. Linares-Vsquez et al. [40] Analysed the mobile development related discussions from SO. They used LDA to extract the main topics that represent those discussions.

Our work applies a similar process, but we used the topic modeling technique for a different purpose and type of information analyzed. Specifically, here, we aim to find the difference in issue report topics across summary classes based on the most frequent topics and their associated probabilities.

VI. Threats to Validity

Selection Bias. Sampling bias is inherent within the limitations of the dataset. The sampling periods consider different time periods across Chromium and Firefox, for which we had to compensate in our analyses. The different time periods are simply due to the fact that, prior to the considered period, Firefox-iOS did not exist. We cannot control any of these variables (team sizes, user bases, etc.). Instead, we can only choose projects that are similar, and the analyzed projects are two large engineering projects with a proper development team behind them.Another sampling threat is that we relied on the

Internal Validity. In this study, the data was collected from Bugzilla for the various distributions of Firefox. For the calculation of issue-fix time in RQ2, only the issues that were marked "fixed" (and, for Firefox, in conjunction with "resolved") at the time of the collection were used in this calculation. However, this may inadvertently be skewed in the future if and when any of these issues may be later reopened. Additionally, we relied on the developers marking the platform that each bug was associated to, i.e., and Android or iOS issue. While there could be mislabeling or issues where such labels might be missed completely, it is out of the scope of this paper to determine if an issue is indeed an Android or iOS issue. Since, we consider bugs that are fixed, we assume that the developers have done their due diligence in marking them correctly. Note that there is no incentive for a developer to mislabel bugs. Finally, in RQ3, we manually interpreted the obtained topics (the topics themselves are automatically derived from LDA) While there was no motivating factor for a bias on our part, we could have incorrectly interpreted the topics. We provide all the topic keywords so that a reader could potentially understand any existing bias.

External Validity. Analyses were done specifically on mobile platform browsers, due to the ease of accessibility and comparatively large datasets. However, there are other cross-platform applications, which are not included in this analysis, which means that there will be other datasets of cross-platform issue reports that should be considered in future work.

VII. Conclusions

In this paper, we reported on an empirical analysis of issues for two web browsers — Chromium and Firefox — on two mobile platforms — iOS and Android. In doing so, we believe that we are the first to study issue reports for two large mobile applications across both major mobile platforms. We found several differences in how issues manifest themselves. We found that the Android versions of both apps had more issues than their iOS counterparts , and we noted that trends of issues also varied. We found that Android issues took longer to fix than iOS in Firefox, but that the opposite was true for Chromium. Furthermore, we found that the majority of Android issues were related to *Content delivery and fragmentation* issues, while iOS issues were more likely to be about *features like tabs and offline use*.

The results of our study suggest that estimating the effort needed for maintaining mobile apps on multiple platforms is difficult, in the sense that the issues that one should expect depend on the platform. Hence, when mobile app developers decide to port their apps to a different platform, they need to be careful not to underestimate the effort needed in the new platform. Our study also opens up several new questions for

the research community and mobile developers, who may wish to obtain a deeper understanding of particular problem areas that are more likely to result in issues on certain platforms. Additionally, practitioners could use our analysis methods as starting point to understand how to allocate resources appropriately to different platforms.

REFERENCES

[1] Schugerl, P., J. Rilling, and P. Charland. "Mining Issue Repositories–A Quality Assessment." In 2008 International Conference on Computational Intelligence for Modelling Control Automation, 1105-10, 2008. doi:10.1109/CIMCA.2008.63.

[2] Bernardi, M. L., G. Canfora, G. A. Di Lucca, M. Di Penta, and D. Distante. "Do Developers Introduce Issues When They Do Not Communicate? The Case of Eclipse and Mozilla." In 2012 16th European Conference on Software Maintenance and Reengineering (CSMR), 139–48, 2012. doi:10.1109/CSMR.2012.24.

[3] Xuan, J., H. Jiang, Z. Ren, and W. Zou. "Developer Prioritization in Issue Repositories." In 2012 34th International Conference on Software Engineering (ICSE), 25–35, 2012. doi:10.1109/ICSE.2012.6227209.

[4] Maji, A. K., K. Hao, S. Sultana, and S. Bagchi. "Characterizing Failures in Mobile OSes: A Case Study with Android and Symbian." In 2010 IEEE 21st International Symposium on Software Reliability Engineering, 249–58, 2010. doi:10.1109/ISSRE.2010.45.

[5] Bhattacharya, P., L. Ulanova, I. Neamtiu, and S. C. Koduru. "An Empirical Analysis of Issue Reports and Issue Fixing in Open Source Android Apps." In 2013 17th European Conference on Software Maintenance and Reengineering (CSMR), 133–43, 2013. doi:10.1109/CSMR.2013.23.

[6] Banerjee, S., J. Helmick, Z. Syed, and B. Cukic. "Eclipse vs. Mozilla: A Comparison of Two Large-Scale Open Source Problem Report Repositories." In 2015 IEEE 16th International Symposium on High Assurance Systems Engineering, 263–70, 2015. doi:10.1109/HASE.2015.45.

[7] Zibran, M. F., F. Z. Eishita, and C. K. Roy. "Useful, But Usable? Factors Affecting the Usability of APIs." In 2011 18th Working Conference on Reverse Engineering, 151–55, 2011. doi:10.1109/WCRE.2011.26.

[8] Ko, Andrew J., and Parmit K. Chilana. "Design, Discussion, and Dissent in Open Issue Reports." In Proceedings of the 2011 iConference, 106–13. iConference '11. New York, NY, USA: ACM, 2011. doi:10.1145/1940761.1940776.

[9] Bettenburg, Nicolas, Sascha Just, Adrian Schroter, Cathrin WeiB, Rahul Premraj, and Thomas Zimmermann. "Quality of Issue Reports in Eclipse." In Proceedings of the 2007 OOPSLA Workshop on Eclipse Technology EXchange, 2125. Eclipse 07. New York, NY, USA: ACM, 2007. doi.org/10.1145/1328279.1328284.

[10] Bettenburg, Nicolas, Sascha Just, Adrian Schroter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. "What Makes a Good Issue Report?" In Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 308–18. SIGSOFT '08/FSE-16. New York, NY, USA: ACM, 2008. doi:10.1145/1453101.1453146.

[11] Bertram, Dane, Amy Voida, Saul Greenberg, and Robert Walker. "Communication, Collaboration, and Issues: The Social Nature of Issue Tracking in Small, Collocated Teams." In Proceedings of the 2010 ACM Conference on Computer Supported Cooperative Work, 291–300. CSCW '10. New York, NY, USA: ACM, 2010. doi:10.1145/1718918.1718972.

[12] Aljedaani, Wajdi, and Yasir Javed. Bug Reports Evolution in Open Source Systems. In 5th International Symposium on Data Mining Applications, 6373. Springer, 2018.

[13] An, L., and F. Khomh. "An Empirical Study of Highly Impactful Issues in Mozilla Projects." In 2015 IEEE International Conference on Software Quality, Reliability and Security (QRS), 262–71, 2015. doi:10.1109/QRS.2015.45.

[14] Nguyen, A. T., T. T. Nguyen, J. Al-Kofahi, H. V. Nguyen, and T. N. Nguyen. "A Topic-Based Approach for Narrowing the Search Space of Issuegy Files from an issue Report." In 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE), 263–72, 2011. doi:10.1109/ASE.2011.6100062.

[15] Thomas, Stephen W., Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. "Modeling the Evolution of Topics in Source Code Histories." In Proceedings of the 8th Working Conference on Mining Software Repositories, 173–82. MSR '11. New York, NY, USA: ACM, 2011. doi:10.1145/1985441.1985467.

[16] Nguyen, A. T., T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun. "Duplicate Issue Report Detection with a Combination of Information Retrieval and Topic Modeling." In 2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), 70–79, 2012. doi:10.1145/2351676.2351687.

[17] Runeson, P., M. Alexandersson, and O. Nyholm. "Detection of Duplicate Defect Reports Using Natural Language Processing." In 29th International Conference on Software Engineering (ICSE'07), 499–510, 2007. doi:10.1109/ICSE.2007.32.

[18] A. Zeller, Why programs fail: a guide to systematic deissueging. Morgan Kaufmann, 2006.

[19] "Mobile Speed Impacts Publisher Revenue - DoubleClick." DoubleClick by Google, September 8, 2016. https://www.doubleclickbygoogle.com/articles/mobile-speed-matters/.

[20] Resnik, P. and Hardisty, E. (2010). Gibbs sampling for the uninitiated. Technical Report UMIACS-TR-2010-04, University of Maryland. http://www.lib.umd.edu/drum/handle/1903/10058.

[21] Goadrich, Mark H., and Michael P. Rogers. "Smart smartphone development: iOS versus Android." Proceedings of the 42nd ACM technical symposium on Computer science education. ACM, 2011.

[22] "Google Chrome - The Fast and Secure Web Browser." Accessed June 21, 2017. https://www.appannie.com/apps/ios/app/chrome-web-browser-by-google/details/.

[23] "Google Chrome: Fast & Secure." Accessed June 21, 2017. https://www.appannie.com/apps/google-play/app/20600000234348/details/.

[24] "Firefox Web Browser." Accessed June 21, 2017. https://www.appannie.com/apps/ios/app/989804926/details/.

[25] "Firefox Browser Fast & Private." Accessed June 21, 2017. https://www.appannie.com/apps/google-play/app/20600000007768/details/.

[26] Zhou, Bo, Iulian Neamtiu, and Rajiv Gupta. "A Cross-Platform Analysis of Issues and Issue-Fixing in Open Source Projects: Desktop vs. Android vs. IOS." In Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, 7:1-7:10. EASE '15. New York, NY, USA: ACM, 2015. doi:10.1145/2745802.2745808.

[27] Resnik, P. and Hardisty, E. (2010). Gibbs sampling for the uninitiated. Technical Report UMIACS-TR-2010-04, University of Maryland. http://www.lib.umd.edu/drum/handle/1903/10058.

[28] Baysal, Olga, Ian Davis, and Michael W. Godfrey. "A Tale of Two Browsers." In Proceedings of the 8th Working Conference on Mining Software Repositories, 238-41. MSR '11. New York, NY, USA: ACM, 2011. doi:10.1145/1985441.1985481.

[29] Rens, Willem. "Browser forensics: adblocker extensions." (2017).Accessed June 16, 2017. http://www.delaat.net/rp/2016-2017/p67/report.pdf.

[30] Uddin, Jamal, Rozaida Ghazali, Mustafa Mat Deris, Rashid Naseem, and Habib Shah. "A Survey on Issue Prioritization." Artif. Intell. Rev. 47, no. 2 (February 2017): 145-80. doi:10.1007/s10462-016-9478-6.

[31] Maguire, Steve. Writing Solid Code: Microsoft's Techniques for Developing Issue-Free Programs. Redmond, WA, USA: Microsoft Press, 1993.

[32] Khomh, F., Dhaliwal, T., Zou, Y. and Adams, B. Do Faster Releases Improve Software Quality? - An Empirical Case Study of Mozilla Firefox, in Proceedings of the 2012 IEEE Working Conference on Mining Software Repositories, MSR (Zurich, Switzerland), pages 179-188.

[33] Israel Herraiz, Emad Shihab, Thanh H. D. Nguyen and Ahmed E. Hassan. Impact of Installation Counts on Perceived Quality: A Case Study on Debian, in Proceedings of the 2011 Working Conference on Reverse Engineering, WCRE (Limerick, Ireland), pages 219-228.

[34] Hu, Hanyang, Shaowei Wang, Cor-Paul Bezemer, and Ahmed E. Hassan. "Studying the Consistency of Star Ratings and Reviews of Popular Free Hybrid Android and IOS Apps." Empirical Software Engineering, 2018, 126.

[35] Hu, Hanyang, Cor-Paul Bezemer, and Ahmed E. Hassan. "Studying the Consistency of Star Ratings and the Complaints in 1 & 2-Star User Reviews for Top Free Cross-Platform Android and IOS Apps." Empirical Software Engineering, 2018, 134.

[36] Ali, Mohamed, Mona Erfani Joorabchi, and Ali Mesbah. "Same App, Different App Stores: A Comparative Study." In Proceedings of the 4th International Conference on Mobile Software Engineering and Systems, 7990. IEEE Press, 2017.

[37] Khalid, Hammad, Emad Shihab, Meiyappan Nagappan, and Ahmed E. Hassan. “What Do Mobile App Users Complain About?” IEEE Software 32, no. 3 (2015): 7077.
[38] Martinez, Matias. “Two Datasets of Questions and Answers for Studying the Development of Cross-Platform Mobile Applications Using Xamarin Framework.” ArXiv Preprint ArXiv:1712.09569, 2017.
[39] Rosen, Christoffer, and Emad Shihab. “What Are Mobile Developers Asking about? A Large Scale Study Using Stack Overflow.” Empirical Software Engineering 21, no. 3 (2016): 11921223.
[40] Linares-Vsquez, Mario, Bogdan Dit, and Denys Poshyvanyk. “An Exploratory Analysis of Mobile Development Issues Using Stack Overflow.” In 2013 10th Working Conference on Mining Software Repositories (MSR), 9396. IEEE, 2013.
[41] Zaman, Shahed, Bram Adams, and Ahmed E. Hassan. “A Qualitative Study on Performance Issues.” In Proceedings of the 9th IEEE Working Conference on Mining Software Repositories, 199–208. MSR ’12. Piscataway, NJ, USA: IEEE Press, 2012. http://dl.acm.org/citation.cfm?id=2664446.2664477.