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

Wajdi M. Aljedaani\textsuperscript{1}, Meiyappan Nagappan\textsuperscript{2}, Bram Adams\textsuperscript{3}, and Michael W. Godfrey\textsuperscript{2}

\textsuperscript{1}Department of Software Engineering, Rochester Institute of Technology, Rochester, NY, USA
\textsuperscript{2}Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada
\textsuperscript{3}MCIS, Polytechnique Montréal, Montréal, Quebec, Canada

\{wma4844\}@rit.edu
\{mei.nagappan, migod\}@uwaterloo.ca
\{bram.adams\}@polymtl.ca

Abstract—In this paper, we compare bugs across two mobile platforms — iOS and Android — for two open source browsers: Mozilla Firefox and Google Chromium. We consider three dimensions: frequency of bug report submission, fixing time of bugs, and type of bugs (using topic modeling on the bug description to generate the categories). Our motivation was to determine if there were differences in how bugs manifest themselves in iOS and Android, when we control for the projects. We found that there were indeed differences. In particular, we found that there were more bugs in the Android apps but with downward trends, compared to upward trends observed for the iOS apps. While we cannot conclusively say which platform requires more time to fix a bug or which type of bugs occur more frequently in a platform, we can conclude that in both apps the fix time and type of bugs are different for each platform. Hence, managers should be more careful in allocating resources to the development on each platform.

Keywords: Bug repository, bug reports; Mozilla Firefox; bug 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: Apples iOS, a closed-source operating system, and Googles Android OS, an open-source operating system. Unsurprisingly, many mobile developers build applications (apps) that target both platforms. Researchers have compared the development of such applications \cite{21}, but to date there has been no study of how bugs for such systems manifest themselves across the two deployment platforms. Indeed, most prior studies on bug repositories have targeted desktop and server applications \cite{18}, as well as apps written for the Android platform \cite{5}. To the best of our knowledge, this is the first empirical study that analyzes bug reports, bug-fixing time, and bug 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 bugs associated to the two most popular 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 bugs across the different platforms, for example whether Firefox on iOS were found to have more usability bugs than Firefox on Android.

Chromium and Firefox were natural choices for this study. As web browsers, they are responsible for the majority of online connectivity via smartphone applications. More importantly, however, they both offer rich, detailed, and freely available bug repositories as datasets. These repositories provide a wide array of supporting data on the types and eventual solutions for the bugs identified by the end users and application developers; comprehensive cross-platform bug datasets such as these are hard to come by.

Specifically, we investigated three research questions in our study:

\textbf{RQ1: How are the numbers of reported bugs distributed across the deployment platforms?}

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

\textbf{RQ2: How are bug-fix times distributed across the deployment platforms?}

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

\textbf{RQ3: How are bug types distributed across the deployment platforms?}

We found that in Firefox, Android bugs took longer to fix than iOS bugs. However, in Chromium, Android bugs took less time to fix than iOS bugs. The longest wait periods before being fixed are in Firefox, while Chromium bugs tend to be finished more quickly. Firefox iOS bugs took less time to fix compared to Chromium iOS.
We performed topic modeling on the textual bug descriptions to infer the underlying types of the bugs; we examined the generated topics and chose labels that seemed to fit best. Using these labels, we found that Firefox Android has more bugs related to Multimedia and Bookmarks, while Firefox iOS has more bugs related to Bookmarks and Post-release Failure. In Chromium Android, more bugs are related to Device-Specific Failure and Crashes, while Pre-release Failures and Third Party Library bugs occur more on the Chromium iOS platform. Hence, there is not a consistent type of bug 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 bugs 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 bug 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 bug tracking systems as well as about the two subject systems of this study.

A. Bug Tracking Systems

Bug tracking system enable the management, tracking, and resolution of programming bugs in large-scale software projects. End users submit their bug 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, and, if actively describing a problem that needs attention, will be addressed by the development team. 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 bug report.

The stages in the lifecycle of a bug report are fairly simple; first, the user submits a report of the symptoms and replication steps of a bug via an online form, which assigns the report for triage (review). Apart from the textual description of a bug, a bug report also records relevant metadata, such as dates, the reporter’s name, and the OS used when the bug occurred. Triggers will then examine and evaluate the bug report to determine if it represents an issue worthy of attention. The outcome could be that the bug is relevant and worth attending to, irrelevant, a duplicate of a known bug, or even actively being worked on already. If it falls under relevant but unattended to, the triager will assign a developer to start addressing the bug report. The developer can interact with the bug reporter and any interested stakeholder through bug report comments. Once finished, the report can be closed if the bug is considered to have been fixed, abandoned if a fix is impossible or impractical, or reopened when an accepted fix is found to be incomplete.

B. Overview of Firefox and Chromium

In this case study, the focus will be on two of the larger open-source bug report repositories that contain bug reports sourced across a variety of platforms. Specifically, the focus is on the Mozilla Firefox and Google Chromium projects, which are the two web browsers that have a majority market share between the main mobile platforms, in particular Android and iOS [20]. We chose the domain of web browsers, since web browsers are known to support multiple platforms and system environments. In the remainder of this section, we now discuss how the Firefox and Chromium projects are being developed, and in which ways this process differs between their Android and iOS versions.

Software Goals. Mozilla Firefox and Google Chromium are widely used web browsers. Mozilla Firefox was released earlier, in 2004, with its codebase originating from a project called Mozilla 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]. It had a market share of 10% in June 2016. Chromium is a newer browser launched in 2008 to facilitate Internet searching. It uses a modified version of the Webkit layout engine also in use by Apple Safari. By June 2016, it had a worldwide market share of 50% [27].

While their main feature sets are comparable, users perceive differences between both browsers 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. Chromium is the default browser on all Android phones1, while Firefox can be downloaded as a separate application.

Source Code. Table I shows the number of source files in Chromium and Firefox for both their Android and iOS version. The Android versions of either app are much larger than their iOS counterparts in terms of number of code files (and also lines of code, not shown here). Furthermore, the Chromium apps reuse much more code from each other (and also lines of code, not shown here). These differences in code reuse might impact the way in which bugs are resolved, as high reuse could mean that one fix could solve bugs in both the Android and iOS version of an app.

Teams. The large amount of reuse in Chromium’s apps can also be noticed by the large number of developers shared by the iOS and Android apps. Both Chromium apps share 859 developers (Table I), on top of which there are 92 Android-only and 12 iOS-only developers. On the other hand, the two

1 It should be noted that a third browser, Safari, is the default for iOS devices; however, it is not a viable candidate for this study as there is no Android version, and much of the development is closed source and thus not amenable to study.
Firefox apps only share 18 developers. These are assisted by 280 Android-only developers and 89 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 bugs are being addressed.

**Release Frequency.** Table II shows the summary of the release cycle times of all 4 apps [22], [23], [24], [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 bugs are being addressed across platforms, since projects with shorter cycle time might have less time for quality assurance [32].

### III. STUDY DESIGN

This section presents the study approach, data collection and data processing of our case study, which aims to address our three research questions.

#### A. Study approach

This section presents the design of our study on the bug repositories of the iOS and Android versions of Firefox and Chrome. Figure 1 shows an overview of our approach. For each case study system, we first extract the necessary data from the entire bug repository. Then, we identify the bug reports related to a specific platform, then calculate relevant metrics for those bug reports. We then analyzed the bug 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 bug reports of Firefox filed for its Android and iOS apps in the Mozilla project’s Bugzilla bug repository\(^2\) for the period from January 01, 2014 to July 23, 2016, yielding 14,082 bugs. The details of our datasets are listed in Table III. In addition, we gathered the data of all the bug reports of Chromium that were submitted to its Monorail issue tracker\(^3\) for the period from March 03, 2015 to July 24, 2016. This yields a total of 5,824 bugs. Note that in our study, we consider all type of issues present in the repository as bugs. This could include features as well. We consider all of them as one (which for consistency we call bugs), as both bugs and features require human effort to address in the app, and in this study the goal is to see if there are differences in the platforms. We also considered any reported bug in our study. In specific RQs we filter out some bugs (for example in RQ2 and 3 we only consider closed bugs).

#### C. Data Processing

First, for each bug repository, we extracted the necessary data from the bug repository (i.e., bug ID, status, summary, OS, reported date and closed date). We then used the R

\(^2\)https://bugzilla.mozilla.org/

\(^3\)https://bugs.chromium.org/hosting/
programming language to perform all analyses for each of the research questions. Although Firefox has recorded its bugs since 2002, Chromium only started in 2015. To be able to make more reliable and accurate comparisons between both software systems, we filtered the Firefox bug report data to include only bugs reported between January 1, 2014, and July 24, 2016.

IV. STUDY RESULTS

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

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

Approach. Collecting the dataset for the Mozilla Firefox-related bugs was quite simple, due to the helpful structure of the Mozilla bug report system; all Firefox bugs tagged with Android and iOS were searched for and downloaded, resulting in a database of 102,063 bug reports. From this dataset, the date range was narrowed, as the entirety of the Mozilla bug report repository dates back to 2002; this dataset was then narrowed to allow for a parallel time period to the Chromium dataset, and to select a time period that was more recent and reliable. Thus, the date range was reduced to January 1, 2014, and July 24, 2016, resulting in a total dataset of 14,082 bug reports, i.e., 10,852 for Android and 3,230 for iOS (cf. Table III).

Collecting the equivalent dataset from the Chromium bug repository was more difficult. As the Chromium bug 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 bugs), it only allowed for the selection of a number of bug 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 bug reports from the time period of interest. Once this selection was complete, the dataset totaled 50,078 total reports. This Chromium dataset was then processed in order to allow for proper cross-platform comparison between the two dataset sources, as follows.

First, Chromium bug reports that contained multiple OS names and related terms in the OS descriptor field were duplicated, with one duplication for each relevant OS, and these reports were then sorted into the appropriate categories. For instance, if the bug report contained the names of Mac, Android, and iOS, in the OS field, then two duplicates would be made of the report, and the three copies would be sorted into the appropriate OS databases. This allows for a proper analysis of bug frequency across the different OS types. The number of reports that were duplicated in this fashion totaled 5,923 “new” reports, which, after this processing and adding to the existing dataset, brought the total number of reports in the dataset to 56,001.

Step two in the processing stage was to remove all bug reports that did not contain a specific OS name or a related term in the relevant field of the bug report. These reports were discarded, as being not relevant to the focus of this research. This included bugs with the “[empty]” descriptor in the OS field. This stage eliminated 13,000 reports from the dataset, bringing the total number of bug reports down from 56,001 to 43,001 reports. Once this stage was complete, the bug reports for Android and iOS were retained, while the other OS-related bug reports were set aside. This collection was then used as the basis of the analytical dataset for Chromium, resulting in a total of 5,152 bug reports for Chromium Android and 674 reports for Chromium iOS.

Results. Figure 2 shows the number of bugs in Firefox for both Android and iOS versions. The first three quarters of 2014, there were a substantial number of bugs experienced on the Android platform. However, as seen in Table III, no bug data exists for the iOS platform for the first three quarters. The results started changing during the fourth quarter of the year where the number of bugs decreases for Android platform, and a small number of bugs are captured in the iOS system. Throughout the four quarters of 2015, there was a varying number of bugs for both Android and iOS platforms. For Android, there was a lesser number noted as compared to the ones in 2014, while the number of bugs on the iOS platform increased in the first three-quarters of 2015. The last quarter of the same year, the number of bugs decreases for both the Android and iOS platform. During the first quarter of 2016, there was a large increase for the Android devices compared to the increase in some bugs noted in the iOS

![Fig. 1: Overview of our approach to extract the metrics of bug reports and analyze the data.](image-url)
platform. The number decreased in the second and the third quarter in both platforms.

For Chromium (Figure 3), we started collecting data from the third quarter of 2015. The number of bugs noted during this quarter shows there were only a few bugs in Chromium for both Android and iOS. The number of bugs increases for the Android platform from 302 to 1,428 in the fourth quarter of the same year, and for the iOS platform from 40 to 221, i.e., both platforms see a 5-fold increase. While in the first three months of 2016 the number of bugs in Android keeps on increasing, the number decreases slightly on the iOS platform. During the second quarter, the number stagnates for both Android and iOS. During the fourth quarter, the number of bugs decreases by a factor of 4.5 on both platforms.

Discussion. The trends that we observed in Figure 3 confirm the large overlap between the iOS and Android teams and code base of Chromium that we saw in Section II, as the number of bugs evolved proportionally, albeit at different levels of magnitude. The reason why the number of bugs is the lowest in the third quarters of 2015 and 2016 is the introduction of new features in Chromium Android related to tabs, security and plugins from the fourth quarter of 2015 to the second of 2016. The changed focus towards bug fixing in the third quarter of 2016 reduced the reported number of bugs for those releases.

In contrast, the much lower overlap in developers and code base between the iOS and Android versions of Firefox led to different trends in Figure 2, especially in the fourth quarter of 2014, the first quarter of 2015 and the fourth quarter of 2015 to the second quarter of 2016. Yet, if one normalizes the number of bugs by the number of files, we obtain a density of 0.5% bugs per file for Firefox Android compared to 0.6% for Firefox iOS (if we use the overall number of files for the 2016 second quarter as reported in Table I).

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. As shown by Herraiz et al. [33], this explains for a large part the different orders of magnitudes in number of reported bugs.

In summary, there are consistently more bugs for the Android platform in both Firefox and Chromium. Whether this is due to the fact that the Android code base of the studied two apps is much larger than their iOS code base, or because the Android platform has more users, we recommend software managers to consider allocating more resources to their Android teams. However, at the same time, due to the increasing trends of bugs observed during some quarters for the iOS app, managers should not underestimate the number of people working on the iOS app either.

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

Approach. The data was sorted and analyzed to determine the time periods taken to fully repair a given bug, from the time it was first opened to the time it was marked as “Resolved” (Firefox) or “Fixed” (Chromium). Bugs that were not marked with either of the two above statuses were not extracted from the major dataset, as they are not part of the dataset being examined for this research question—the focus is on bugs that have been repaired. Once this selection was complete, the dataset that was examined for analyzing the fixing time for Firefox Android is 2,670 and for Firefox iOS 1,700, while for Chromium Android it is 1,775 and Chromium iOS 218. The bug fixing-time is calculated as the number of days between opening and closing of a bug report.

We then used a Wilcoxon statistical analysis and pairwise statistical analysis on the data subsets to determine which browser/platform combination possesses the largest respectively lowest times to repair.

Results. Figure 4 shows a box-plot of the Firefox bug resolution in number of days. The median number of days taken to fix bugs was higher for Android compared to iOS. On average it takes a median of 12 days to fix a bug on Android compared to 8 (see Table IV). Out of 2,670 bugs 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
Table IV: Summary of fixing time of the projects (in #days).

<table>
<thead>
<tr>
<th>Company</th>
<th>Platform</th>
<th>Min.</th>
<th>Q1</th>
<th>Median</th>
<th>Mean</th>
<th>Q3</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>Android</td>
<td>0.00</td>
<td>3.25</td>
<td>12.00</td>
<td>42.95</td>
<td>37.00</td>
<td>800.00</td>
</tr>
<tr>
<td></td>
<td>iOS</td>
<td>0.00</td>
<td>2.00</td>
<td>8.00</td>
<td>31.02</td>
<td>33.00</td>
<td>390.00</td>
</tr>
<tr>
<td>Chromium</td>
<td>Android</td>
<td>0.00</td>
<td>2.00</td>
<td>10.00</td>
<td>29.87</td>
<td>34.00</td>
<td>295.00</td>
</tr>
<tr>
<td></td>
<td>iOS</td>
<td>0.00</td>
<td>3.25</td>
<td>20.00</td>
<td>40.77</td>
<td>56.75</td>
<td>251.00</td>
</tr>
</tbody>
</table>

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

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

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

<table>
<thead>
<tr>
<th>Company</th>
<th>X vs. Y</th>
<th>Less</th>
<th>Greater</th>
<th>2-sided</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firefox</td>
<td>Android vs. iOS</td>
<td>1</td>
<td>1.852e-11</td>
<td>3.704e-11</td>
</tr>
<tr>
<td>Chromium</td>
<td>Android vs. iOS</td>
<td>5.146e-05</td>
<td>0.9999</td>
<td>0.0001029</td>
</tr>
</tbody>
</table>

Discussion. We performed a Wilcoxon Rank-Sum hypothesis test to understand whether the observed differences in the distribution of bug fixing time are significant. From the results in Table V, we can see that in Firefox, Android bugs took a significantly longer time to fix than iOS bugs. However, in Chromium, Android bugs took significantly less time to fix than iOS bugs. Given that the Android apps both have a similar much shorter release cycle time than their iOS apps, it is surprising that Firefox’ median bug fixing time on iOS is in fact lower than the time for Android! Deeper analysis is needed to better understand this. For example, although it is known that a large part of the release cycle can be taken up by the iOS app store reviewing a new release rather than by actual development, a similar effect should be seen for Chrome, which is not the case. In any case, we cannot conclusively say whether Android bugs take more or less time to fix as compared to iOS bugs, i.e., both platforms can be associated with longer bug fixing times.

Despite this observation, one aspect 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 bugs will be simple repairs that can be taken care of in a very short period of time, meaning that the gradually descending curve is more representative of developer bottleneck than of actual time taken to repair the bug. The results still indicate that there is room for improvement in the various bug-repair development processes; while there are a number of bugs that are difficult to fix, they are a minority in comparison with the fairly basic yet far more numerous bugs that eat up the majority of the developers’ attention resources.

RQ3: How are bug types distributed across the deployment platforms?

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

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

We selected randomly 3K bugs from Chromium and 3K from Firefox to perform the LDA analysis, since this represents a sample with a 99% confidence level and a 2% confidence interval 4.

4https://www.surveysystem.com/sscalc.htm
Results. Tables VI and VII show the 10 topics and the keywords that LDA selected as most representative of the topics. Figures 6 and 7 show the frequency of bugs in each of the 10 topics. Below we will discuss the top two topics with the largest differences between the two platforms of Firefox and Chromium.

From Figure 6 we can see that topics 3 and 8 of Firefox have substantially more bugs on Android than on iOS. Topic 3 covers 10.65% of all Android bugs compared to 8.23% of all iOS bugs. From the keywords for topic 3 in Table VI, we can see that the topic’s corresponding bugs are “multimedia” related bugs. Manual analysis of these Android bugs show that they are mostly related to the JWPlayer not displaying video content in full screen. Topic 8 on the other hand covers 11.91% of all Android bugs and 8.09% of all iOS bugs. From the keywords in Table VI (and manually examining the bugs related to topic 8), we can see that topic 8 is mostly related to “bookmark links” in the browser.

Conversely, from Figure 6, we can also see that the iOS platform has a larger share of bugs in topics 7 and 9 compared to Android. Topic 7 covers 11.52% of all iOS bugs for Firefox compared to 8.92% of all Android bugs. From the keywords in Table VI, it seems like these issues are related to bookmarks as well. However, on closer examination, we find that most of these bugs are post-release failures related to bookmark as a feature. Similarly, we can see that topic 9 is related to post-release failures of specific builds.

Figure 7 shows the bug distribution across the iOS and Android platforms for the 10 topics in Chromium. For the Android platform, topics 2 and 8 have more bugs in Android than in iOS (with the difference being the largest among the 10 topics). Topic 2 has 11.49% of all bugs in Android while the iOS platform just has 7.64% of the iOS bugs. From Table VII, we can see that topic 8 is related to “device specific failures”. By examining the bugs in topic 2 manually, we found that the bugs were related to specific devices like the Nexus phone. Topic 8 has 11.19% of the Android bugs compared to only 6.46% of the iOS bugs. From Table VII we can see that topic 8 is related to “crashes”.

Conversely, from Figure 7 we can see that the topics 3 and 7 have considerably more bugs in iOS than Android. In topic 3, the difference is slightly under 3%, while in topic 7 the difference is almost 7%. From Table VII, we can see that topic 3 is related to “pre-release failures” that occurred in the testing phase. In other words, these bugs are reported by developers rather than users. From Table VII, we can see that topic 7 is about “third party libraries”. By examining the bugs associated with this topic, we found that most bugs were related to xcode (the IDE used to develop iOS apps).

Discussion. As stated earlier, this study selected 3K bugs from Android and iOS for LDA analysis. The identified bug types vary from minor ones like a video not playing or bookmark not being added to crucial types like security bugs or memory leaks. From examining the above results, Firefox’s major issues arise from end-user issues, while Chromium, in contrast, is very nearly evenly split between end-user and development issues. For both browsers, this is a general trend observed between both Android and iOS platforms.

For Firefox, if the page refuses to load or the browser bar fails to execute a search, the user can and will get disgruntled rapidly, since prior research has shown that a web page being slower to load will result in people abandoning it, with 53% of mobile web pages being abandoned after three seconds of waiting [19]. From this, it is easy to extrapolate that end users will be extremely sensitive to inconsistencies and reduced functionality in their browsers.

While these issues, i.e., the program not doing what the user wants it to do, form the essence of problems reported for Firefox, for Android Chromium bugs exist not only because of feature-related issues, but also because of the developers inadvertently introducing an issue in the code. Why this is would be mere speculation, but, knowing that Chromium is one of the default browsers on all Android devices and is very popular in its own right, it may be that Chromium is a victim of its success [33], with the larger popularity translating into more issues on device-specific failures.

In conclusion, while there is not a specific type of bug that is prevalent in either the iOS or Android platforms, app developers could use topic analysis to determine if
a specific type of bug is more likely to be present in one platform over the other and recruit experts accordingly.

V. RELATED WORK

Bug repositories have become a vital database in software development and maintenance of large-scale systems. They provide crucial information to developers and permit users to inform developers of the issues that the users experienced while utilizing the software. There are many studies that are mainly related to bug repositories because of their significance. Furthermore, most of the papers related to bug repositories are targeted to relating the level of communication amongst commuters to bug-proneness [1], evaluation of bug 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 bug repositories. We discuss three areas of related work and overlaps, as well as present the differences between the related work and our work.

A. Mobile Bugs Studies

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

In a study similar to ours, Bhattacharya et al. [5] performed an empirical study on bugs 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 bug reports, bug fixes, and security bugs. Their approach for measuring the bug-fixing time was structured as follows: they subtracted the time taken from the instant where the bug was reported and to the moment where the bug closed to get the actual length of time it took to resolve the bug in months. They found that, for the majority of the apps, the average number taken to fix bugs ranges from 0 to 1.5 months.

Our approach is similar to theirs, but the difference is that instead of concentrating only on bugs on the Android platform, we are also focusing on bugs on the iOS mobile platform. Furthermore, we performed our case study on both Firefox Mozilla’s and Google Chromium’s bug reports. This indicates that the dataset that we used for this study is larger and more varied than the one contained in prior studies.

B. Empirical bug studies and Qualitative Analysis

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

Banerjee et al. [6] compared two large open-source bug 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; the only difference is that instead of focusing on Eclipse and Mozilla, we concentrated on Mozilla and Chromium. We examined the evolution of bug repositories over time by analyzing the number of submitted report per year.

A few studies have performed the qualitative analysis of bug repositories using a survey, questionnaires, and open-source projects [7], [8], [9], [10], [11]. An et al. [12] examined the characteristics of highly impactful bugs in Mozilla Firefox and Fennec Android. They proposed statistical models to assist the software organizations to predict the highly impactful bugs early before affecting a large number

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>tab</td>
<td>menu</td>
<td>crash</td>
<td>page</td>
<td>mode</td>
<td>search</td>
<td>sync</td>
<td>add</td>
<td>firefox</td>
<td>url</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>open</td>
<td>doesn't</td>
<td>display</td>
<td>android</td>
<td>view</td>
<td>site</td>
<td>bookmark</td>
<td>list</td>
<td>ios</td>
<td>bar</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>private</td>
<td>toolbar</td>
<td>screen</td>
<td>load</td>
<td>button</td>
<td>local</td>
<td>panel</td>
<td>read</td>
<td>build</td>
<td>text</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>close</td>
<td>string</td>
<td>icon</td>
<td>content</td>
<td>reader</td>
<td>use</td>
<td>history</td>
<td>action</td>
<td>support</td>
<td>show</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>brow</td>
<td>share</td>
<td>update</td>
<td>app</td>
<td>change</td>
<td>default</td>
<td>fail</td>
<td>allow</td>
<td>data</td>
<td>update</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>browser</td>
<td>option</td>
<td>video</td>
<td>create</td>
<td>state</td>
<td>suggest</td>
<td>user</td>
<td>item</td>
<td>save</td>
<td>dont</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE VI:** Summary of the terms in Firefox for 10 topics.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>chrome</td>
<td>android</td>
<td>test</td>
<td>video</td>
<td>text</td>
<td>page</td>
<td>ios</td>
<td>android</td>
<td>add</td>
<td>scroll</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>tab</td>
<td>fail</td>
<td>web</td>
<td>work</td>
<td>client</td>
<td>offline</td>
<td>implement</td>
<td>chrome</td>
<td>webview</td>
<td>mode</td>
<td>create</td>
</tr>
<tr>
<td>3</td>
<td>open</td>
<td>device</td>
<td>run</td>
<td>doesnt</td>
<td>version</td>
<td>load</td>
<td>chrome</td>
<td>webview</td>
<td>mode</td>
<td>create</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>app</td>
<td>build</td>
<td>disable</td>
<td>remove</td>
<td>update</td>
<td>save</td>
<td>api</td>
<td>site</td>
<td>enable</td>
<td>screen</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>link</td>
<td>perf</td>
<td>call</td>
<td>change</td>
<td>browser</td>
<td>url</td>
<td>request</td>
<td>system</td>
<td>start</td>
<td>screen</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>download</td>
<td>bot</td>
<td>dont</td>
<td>play</td>
<td>blimp</td>
<td>bookmark</td>
<td>notif</td>
<td>tri</td>
<td>display</td>
<td>access</td>
<td></td>
</tr>
</tbody>
</table>

**TABLE VII:** Summary of the terms in Chromium for 10 topics.
of users. Furthermore, they compared the bug fix rate of highly-impactful bugs vs. other bugs. 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. [13] studied bugs in the desktop versions of Mozilla Firefox and Google Chromium to observe the collaboration among project members in terms of detecting and fixing performance bugs. Their approach performed a comparison on a random sample of 400 performance and non-performance bug 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 there should be more analysis of the impact of changes in performance.

Our work is different from the latter work in both purpose and approach. Our purpose is to compare bug reports across two mobile platforms instead of only between two projects. Moreover, we focused on the analysis of bug fixing-time to identify if these bug repositories can help Firefox and Chromium to improve the bug-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], bug localization [15], and duplicate bug detection [16], [17]. Our work applies a similar process, but we used the topic modeling technique for a different purpose. Specifically, here, we aim to find the difference in bug 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 prior to the consider period, Firefox iOS did not exist.

Internal Validity. In this study, the data was collected from Bugzilla for the various distributions of Firefox. For the calculation of bug-fix time in RQ2, only the bugs 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 bugs may be later reopened. Additionally, in RQ3, we manually interpreted the obtained topics. 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 bug reports that should be considered in future work.

VII. CONCLUSIONS

In this paper, we reported on an empirical analysis of bugs 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 bug reports for two large mobile applications across both major mobile platforms. We found several differences in how bugs manifest themselves. We found that the Android versions of both apps had more bugs than their iOS counterparts, and we noted that trends of bugginess also varied. We found that Android bugs took longer to fix than iOS in Firefox, but that the opposite was true for Chromium. Furthermore, we found that the majority of Firefox bugs related to End-User Issues, while Chromium bugs were more evenly split between End-User Issues and Development Issues.

While we cannot conclusively say which platform requires more time to fix a bug or which type of bugs occur frequently in a platform, we can conclude that in both apps the fix time and type of bugs are different for each platform. Porting and maintaining mobile apps on multiple platforms is not a trivial activity! Therefore, we feel that this study opens up a wide variety of questions for mobile developers, who may wish to allocate their development resources according to particular problem areas that are more likely to result in bugs. Managers could use our analysis methods to allocate such resources appropriately.

REFERENCES


