Empirical Investigation of Accessibility Bug Reports in Mobile Platforms: A Chromium Case Study

Wajdi Aljedaani
wajdi.j1@gmail.com
University of North Texas
USA

Mohamed Wiem Mkaouer
mwmvse@rit.edu
Rochester Institute of Technology
USA

Marcelo M. Eler
marceloeler@usp.br
University of São Paulo
Brazil

Marouane Kessentini
marouane@umich.edu
University of Michigan-Flint
USA

Abstract

Accessibility is an important quality factor of mobile applications. Many studies have shown that, despite the availability of many resources to guide the development of accessible software, most apps and web applications contain many accessibility issues. Some researchers surveyed professionals and organizations to understand the lack of accessibility during software development, but few studies have investigated how developers and organizations respond to accessibility bug reports. Therefore, this paper analyzes accessibility bug reports posted in the Chromium repository to understand how developers and organizations handle them. More specifically, we want to determine the frequency of accessibility bug reports over time, the time-to-fix compared to traditional bug reports (e.g., functional bugs), and the types of accessibility barriers reported. Results show that the frequency of accessibility reports has increased over the years, and accessibility bugs take longer to be fixed, as they tend to be given low priority.

CCS Concepts

• **Human-centered computing** → **Empirical studies in accessibility**; **Ubiquitous and mobile devices**.

Keywords

Bug Report, Accessibility, Bug Repository, Google Chromium, Mobile applications, Open Source, Empirical Studies.

ACM Reference Format:
Wajdi Aljedaani, Mohamed Wiem Mkaouer, Marcelo M. Eler, and Marouane Kessentini. 2024. Empirical Investigation of Accessibility Bug Reports in Mobile Platforms: A Chromium Case Study. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24), May 11–16, 2024, Honolulu, HI, USA.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3613904.3642508

CHI '24, May 11–16, 2024, Honolulu, HI, USA

ACM ISBN 979-8-4007-0330-0/24/05
https://doi.org/10.1145/3613904.3642508

1 Introduction

Mobile applications have become ubiquitous in the modern age as they are used by billions of people every day. Digital accessibility plays a crucial role in this context, as it is intended to ensure that digital products, services, and content can be perceived, understood, and operated by a wide range of users, regardless of their physical, intellectual, or mental capacity [57]. Unfortunately, despite the existence of an extensive body of knowledge and tools to support the development of accessible software, many studies have reported that most mobile and web applications still impose several barriers to people with disabilities [13, 24, 45, 61].

Consequently, some researchers investigated the perceptions and opinions of developers when it comes to their awareness and knowledge of accessibility, as well as the main barriers and motivations regarding the inclusion of accessibility requirements in the development process [21, 39]. In general, participants reported that the main barriers are the lack of knowledge and training, as well as the lack of accessibility requirements from each project or organization. Bi and Xia [20] investigated accessibility issues reported by developers in popular GitHub projects over two years. The authors collected 11,820 accessibility issues and grouped them into seven categories based on the type of issue. The study revealed the potential causes of accessibility issues and the lack of knowledge among developers about these issues. However, the study was conducted over a single period of time, so it is not clear if the findings are generalizable to other time periods.

The existing literature provides insights into why accessibility is often not addressed during the development process. However, there is a paucity of studies that examine how developers and organizations respond when evidence of the lack of accessibility of their software is presented. For example, few studies have investigated how developers and organizations address accessibility bug reports [35], how they are classified [12], and how long it takes to fix them [64].

Accordingly, we present an investigation whose purpose is to study the accessibility bug reports on the repository of the mobile version of Chromium in order to understand how developers and organizations handle them in comparison with traditional bug reports (e.g., crashes, functional bugs). More specifically, we want to identify i) the distribution of accessibility bugs reported over time; ii) how developers and organizations prioritize bug reports, including the reasoning behind their choices and the time they take

to fix them; and iii) what is the cause of most accessibility issues reported? We chose to study bug reports for Google Chromium because it is a widely used open-source project behind the Google Chrome browser and the Google Chrome OS. In addition, this repository is easily accessible and offers adequate data considering the scope of our study. To guide our investigation, we framed our study around the following research questions:

RQ_1: What is the progression of accessibility bug reports?

With this RQ, we want to understand whether the number of bugs being reported is increasing or decreasing over time, which can be an indication of the accessibility evolution of the Chromium mobile app and of the user involvement in reporting barriers to improve Chromium's accessibility. We conduct such analysis for both Android and iOS platforms. We found that the Android browser app has more accessibility issues than iOS, and the number of issues is increasing over time, whereas the iOS accessibility issues have reduced over time.

RQ_2: How does the time-to-fix of accessibility bug reports compares to the time-to-fix of other bug reports?

With this RQ, we want to understand how accessibility bug reports are handled in practice, considering how much time developers take to fix both accessibility-related and non-accessibility related bugs reported. In addition, we investigated how accessibility bugs are classified and prioritized in the bug-fixing cycle. Such information can give us insight into whether developers acknowledge or dismiss accessibility as important as any other quality feature. We found that fixing accessibility issues takes longer and is given less priority than fixing non-accessibility issues, which are given high priority regardless of the mobile platform.

RQ_3: What is the root cause of accessibility bug reports?

With this RQ, we want to identify the specific accessibility violations that led to each accessibility bug report. A thorough categorization of accessibility-related bug reports was conducted on the Android and iOS platforms to address this RQ. Two reviewers carefully read and classified each accessibility bug report. By understanding the most common accessibility issues highlighted in these bug reports, developers can proactively address them in their current and future applications, ensuring improved accessibility for a wider range of users.

This paper is organized as follows. Section 2 presents some background information related to bug-tracking systems and mobile accessibility. Section 3 describes our study design, while Section 4 presents and discusses the results by answering our research questions. Section 6 outlines the related work. The threats to the validity of our study are presented in Section 7. Finally, Section 8 presents some concluding remarks and future directions.

2 Background

In this section, we present some background information about bug-tracking systems, Chromium software systems, and mobile accessibility guidelines.

2.1 Bug Tracking Systems

A bug tracking system, also known as an issue tracking system, enables managing, tracking, and resolving programming issues in wide-ranging software projects. The software in use frequently auto-generates the end user's problem in the form of a report that is then examined and fixed by developers. These reports are collected, checked, and screened; if a noticeable problem is found in the report, it will be assigned to a developer to resolve the issue. Once addressed, these reports are documented and cataloged accordingly so that if their need arises in the circumstance of a later case, they can be referred to reliably. The issue reports' lifecycle is quite simple: firstly, the symptoms and replication steps are submitted by the user in a report through an online form, and it is allocated for review (triage).

2.2 Overview of Chromium

In this case study, we focus on one of the largest open-source bug report repositories, Chromium platforms, which is spread across various mobile platforms. As the Chromium application is supported on various platforms and system environments, we chose the domain of web browsers. Specifically, we chose Google Chromium's Android and iOS, as these are open-source projects that are widely used and whose development methods can be tracked and analyzed by anybody. In the rest of this section, we discuss how the Chromium projects are evolving and how this process varies between the Android and iOS versions. We also provide guidelines for mobile accessibility.

Software Goals. Google Chromium is a widely utilized web browser in the digital domain. Launched initially as a desktop version in 2008, its first mobile iteration was introduced in 2011. Similarly to other contemporary browsers such as Safari, Chromium is constructed upon the WebKit framework. It is essential to differentiate between two variants of Google browsers: Chrome and Chromium. Chrome, a branded version by Google, caters to the majority of end-users as a closed-source project. Nevertheless, it is predicated upon its open-source counterpart, Chromium, which is developed and disseminated under a BSD license. Google's Chrome enhances Chromium with an array of exclusive features, including support for diverse media formats. Additional benefits of Chromium encompass the absence of usage data storage and transmission, as well as the capacity to disable the security sandbox on Linux platforms. Due to the inaccessibility of data from closed-source components of Chrome, this project has opted to study Chromium as a proxy for Chrome. Considering that a significant portion of Chrome's strategic functionality is derived from Chromium, we consider it a reasonable choice.

While the primary feature sets of Android and iOS Chromium are similar, users may notice differences in aspects such as battery consumption, accessibility, startup time, security/privacy (a Google-backed project versus an open-source project [49]), plugin availability, and more. Chromium provides a mobile version of its application for both Android and iOS platforms. In 2013, the release of Android version 4.4 ("KitKat") marked Chrome as the default browser for the Android platform, while Safari remained the default browser for the iOS platform. Users can typically download and install these apps independently if desired. Although numerous iOS

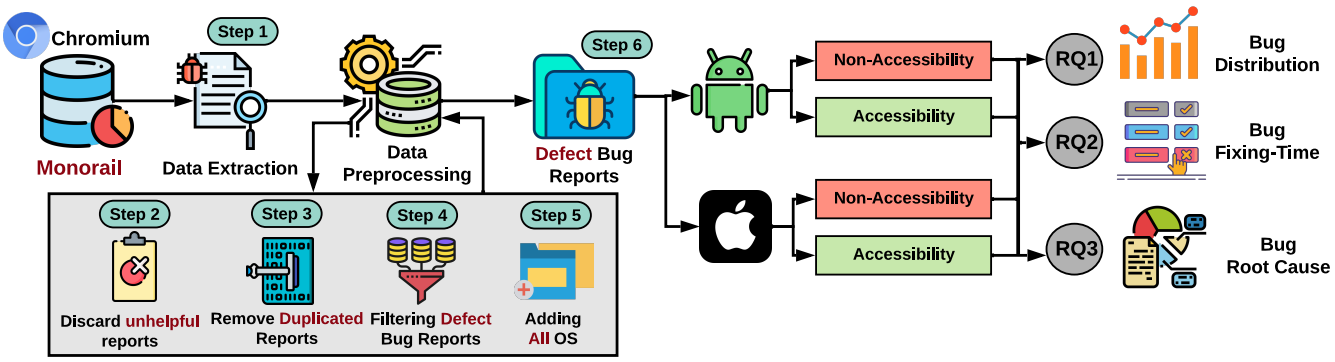

Figure 1: Overview of our study approach.

users express interest in using the Chromium app, some opt for Chrome instead since Safari is primarily a closed-source project that has not been ported to Android and, thus, has not been examined in this study.

Release Frequency. Table 1 summarizes the release cycle duration for both iOS and Android applications [37, 47, 48], representing the number of days between consecutive official releases. This table does not distinguish between major and minor versions and solely includes official releases accessible to customers through official app stores, excluding any unofficial releases, such as beta versions. The table reveals Chromium's median release cycle times for Android (one-week median) and iOS (three-week median). This observation suggests that the median release cycle times for iOS are twice as long as those for their Android counterparts. These disparities may highlight differences in issue management across platforms, as projects with shorter cycle times might have reduced time allocated for quality assurance [38].

Table 1: Summary of the time between consecutive releases (in #days) for the two platforms.

Platform	Min.	Q1	Median	Mean	Q3	Max.
Android	1.00	3.00	7.00	14.04	17.00	103.00
iOS	1.0	10.5	21.0	24.8	37.0	77.0

2.3 Mobile Accessibility Guidelines

Mobile accessibility is addressed by both WCAG 2.2 and the U.S. Revised Section 508 standards. For web content, the primary focus is on WCAG 2.2 [57]. To apply WCAG 2.2 to mobile applications and content, W3C provides specific guidance [58]. The U.S. Revised Section 508 standards mandate that mobile content complies with WCAG [22]. Additional regulations exist, such as the BBC Standards and Guidelines for Mobile Accessibility from the UK [15], which presents a series of easily implemented best practices for mobile content. The IBM Accessibility Checklist [1] consolidates the U.S. Revised Section 508 standards and WCAG 2.0 Level A and Level AA Success Criteria (SC) into checkpoints, each representing a distinct accessibility requirement. Since the accessibility requirements of the U.S. Revised Section 508 and WCAG 2.0 differ, the IBM checklist encompasses both to cater to customers from various countries and regions. For example, customers in Europe often require WCAG 2.0, whereas those in the United States may need U.S. Section 508. The IBM Checklist is flexible, allowing customers to select both U.S. Revised Section 508 and WCAG 2.0 for their web pages or mobile applications or to utilize just one of these.

In addition to these standards, mobile device manufacturers offer platform-specific guidelines, such as Android User Interface Guidelines [29], Google's Accessibility Developer Checklist [28, 31], and Apple's iOS Human Interface Guidelines for Accessibility [30]. These guidelines supply information on platform-specific accessibility infrastructure and APIs, which subsequently facilitates the mapping of accessibility requirements to platform implementations. IBM Mobile Accessibility Checker (MAC) [2] employed specific guidelines for iOS and Android platforms to establish accessibility rules and rule mappings for corresponding widgets.

3 Study Design

This section explains the methodology of our research and how we gathered and analyzed the data to answer our research questions.

3.1 Study Approach

In this section, we present our overall study approach to the bug report of the Chromium project on Android and iOS platforms. Figure 1 shows the overall design of our study. First, we extracted all the necessary data from the Chromium bug-tracking system. Chromium uses a Monorail[1] bug tracking system as their primary database to record their bug reports. Monorail is an issue-tracking system in which users can basically request new features and report issues of any type, including bug reports of any type. For each type, there is a template of the information that the user should provide so the developers can respond efficiently or take some action. When it comes to bug reports, for instance, users are suggested to write the steps that will reproduce the problem and report the expected and the actual output, in addition to the consequences of the bug being reported along with any additional information.

We then filtered the downloaded data based on the type of the bug report as *Defect* or *Bug*. Next, we identify the bug report based on Android and iOS platforms. For each platform, we split the data extracted based on two types, accessibility-related and non-accessibility-related. We can then define problems relating to a given platform and measure the appropriate metrics for these issues

[1]https://bugs.chromium.org/hosting/

Table 2: Present the process of the data collection steps and the summary of the statistic of datasets.

Chromium Projects	Step (1) # downloaded	Step (2) # Discarded	Step (3) # After cleaning	Step (4) # Defect	Step (5) # Adding All OS	Step (6) # Total	Filtering Report Type	Distribution # Bug Report	Time Spin Start Date	Time Spin End Date
Android	39,633	3,144	36,489	29,503	763	30,266	**Non-Accessibility**	28,958	07-05-2014	19-11-2020
							Accessibility	1,308	03-09-2008	23-11-2020
iOS	11,395	160	11,235	8,305	636	8,941	**Non-Accessibility**	8,092	14-09-2009	17-11-2020
							Accessibility	849	03-09-2008	16-11-2020

reports. Next, we analyzed the problem bug report metrics to define, for each app, potential differences between accessibility bug reports in Android and iOS platforms. The primary metrics examined in this study are defined in-depth in the approach to each of our three research questions.

3.2 Data Collection and Processing

Data collection is the step (1) in our study. Our goal is to study the accessibility bug reports in Chromium mobile platforms. To do so, we collected all bug reports that were archived in the Monorail repository. After downloading the bug reports related to Android and iOS projects from the Monorail database, we performed data cleaning in two steps. In step (2), we removed any bug reports that were unhelpful, testing, or debugging. In step (3), we removed all duplicate bug reports on both platforms. We found that 3,144 and 160 bug reports were removed in these steps for the Android and iOS platforms, respectively. This reduced the number of bug reports to 36,489 in Android and 11,235 in iOS. Furthermore, we only considered bug reports that were typed as *Defect* or *Bug*. We eliminated all other types, such as feature, task, and enhancement, in step (4). This yielded a total of 29,503 bug reports in Android and 8,305 in iOS. We provided some examples in Table 3. While we were mining the bug reports for the selected platforms, we noticed that several bug reports were reported as "All" in the OS field. This means that the bug could affect both Android and iOS platforms. In step (5), we decided to consider each of these bug reports as two separate reports, one for each platform. This is because we wanted to ensure that we captured all of the relevant data for each bug. Therefore, we added 763 bug reports to the Android dataset and 636 bug reports to the iOS dataset. After all five steps, we finalized our final dataset, which yielded a total of 30,266 bug reports for Android and 8,940 bug reports for iOS in step (6). Table 2 summarizes all the data collection and processing steps.

As the bug track system (Monorail) is used to file any type of bugs, after finalizing the complete datasets for both platforms, we needed to distinguish between accessibility and non-accessibility bug reports. In the Chromium bug repository, project teams assign labels to bug reports based on their relevance. To ensure that our datasets of accessibility and non-accessibility bug reports were reliable and accurate, we considered bug reports labeled as "A11y", "accessibility", or "team-accessibility" to be accessibility-related. This systematic approach ensured the validity of our datasets and prevented any potential bias in categorizing the bug reports. In contrast, in traditional bug report repositories, such as Bugzilla, bug reports are not pre-labeled with accessibility tags. To maintain consistency and avoid bias in our study dataset, we ensured that the bug reports in these repositories were appropriately labeled by developers, following a similar criterion used in the Chromium repository for identifying accessibility-related issues.

Table 3: Examples of discarded bug reports.

Type	Description
Non-English	Girdiğim eğitim sitesi güvenlik hatası veriyor
Testing	testing a bug
Non-Meaningful	afdfsadsdsad
Thanking	Thank you!

4 Study Results & Analysis

The findings of our study are presented and discussed in this section. For each of our RQs, we expose the research question, the study approach, and the RQ findings, and then we discuss the observations.

RQ_1: What is the progression of accessibility bug reports?

Motivation. Our main objective with this research question is to analyze whether the number of accessibility-related bugs is increasing or decreasing over time. By acquiring this knowledge, we will be able to gather information on the presence of accessibility barriers on Chromium and possibly the improvement or the decline of Chromium's accessibility. In addition, by understanding the rate at which accessibility bugs appear in both platforms, Android and iOS, it is possible to comprehend the resources possibly required for developers and testers on each platform.

Approach. We downloaded all the bug reports related to iOS and Android from the Monorail database, which contains accessibility-related and other types of bugs. Next, we filtered bug reports based on accessibility and non-accessibility, where the Chromium archive provides a bug report label. If the bug reports are labeled as Accessibility, A11y, or accessibility-team, we consider these bugs as accessibility-related. Otherwise, we label the bug reports as non-accessibility bug reports.

Table 2 presents the summary of the statistics of datasets for each step. After that, we filtered the completed dataset for both accessibility and non-accessibility based on two types of bug reports, Defect or Bug, and all other bug report types such as task, feature, documentation, or "[Empty]" were discarded as not relevant to the study. In the analysis, a total of 30,266 Android bug reports were classified into two distinct categories: 1,308 accessibility-related bug reports and 28,958 non-accessibility-related bug reports. Similarly, the categorization of 8,941 iOS bug reports yielded 849 accessibility-related bug reports and 8,092 non-accessibility-related bug reports.

Results. Figure 2 shows the distribution of accessibility bug reports in Android and iOS platforms from 2008 to 2020. In 2008, a relatively small number of accessibility bugs were reported, i.e., 16 for Android Accessibility and 15 for iOS Accessibility. The bug count for both platforms increased to more than double in the next year, and a similar trend was reported in 2010 and 2011. There was a slight increase in Android Accessibility bugs in 2012. However, the number of iOS accessibility bugs remained constant. In 2013, a

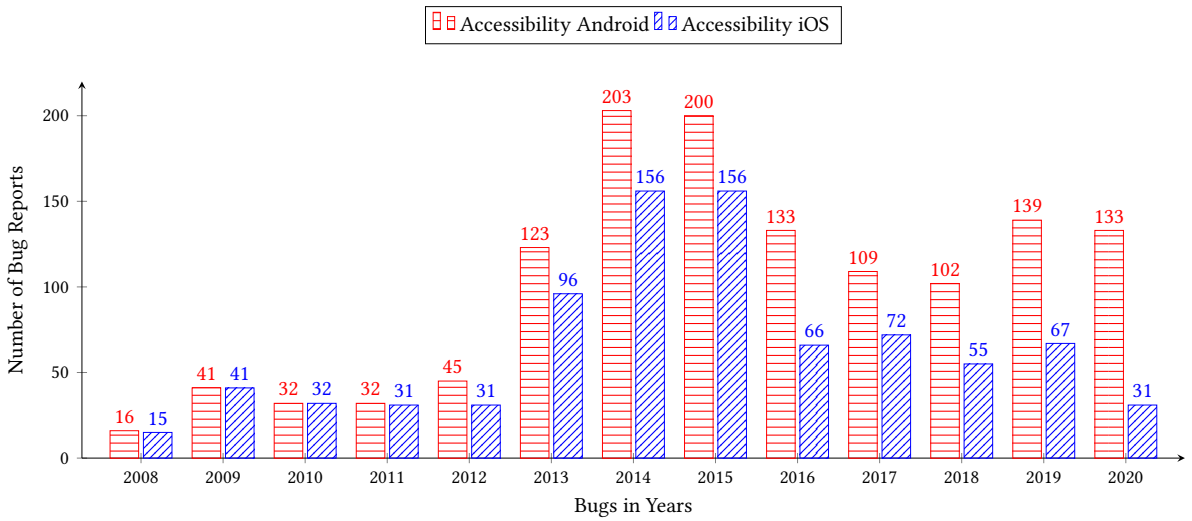

Figure 2: Distribution of accessibility bug reports across in Android and iOS over the years.

drastic increase in accessibility bug reports for Android, as well as iOS, was recorded. The highest number of accessibility bugs were reported in 2014 and 2015. In 2016, there was a slight reduction in accessibility bug reports for Android, but iOS Accessibility bugs were reduced to more than 40%.

During 2017-2018, the number of accessibility bug reports for Android and iOS was more or less similar, with small fluctuations. There was an increase in Android-related bugs in 2019 and 2020. However, iOS-related bugs were reduced. Overall, the number of issues reported for iOS is considerably lower as compared to Android, and it is notable that after 2015, substantial work has been done to control the accessibility-related issues in iOS. It can be seen from Figure 3 that 96% of the reported bugs are related to non-accessibility while only 4% are related to accessibility. Out of these 4% accessibility bug reports, 3% counts for Android while only 1% are related to the iOS platform.

Discussion. Using datasets from the Chromium bug archive, we were able to analyze the progression of the number of accessibility bug reports for Android and iOS, and we discovered that Android had more overall bug reports than iOS. Since the discrepancy in the frequency of bug reports may have ramifications for how issues are resolved, it is interesting to compare the Android and iOS systems in this study.

The patterns shown in Figure 2 support the changing nature of bug reports as the number of issues increased and decreased over time. The number of reported bugs does not necessarily indicate an upward trend in the number of bugs over time; instead, it could be because there has been a considerable growth in the number of accessibility users since 2013, which has resulted in the community becoming more adept at reporting bugs. Our claim is supported by the result of accessibility bug reports, where the highest accessibility-related bugs were reported for both Android and iOS systems in 2014 and 2015. However, until 2018, the trend shifted downward with each passing year, until increasing once more in 2019 for Android.

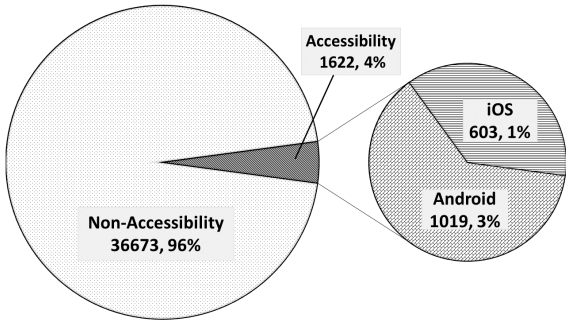

Figure 3: Distribution of accessibility and non-accessibility bug reports across mobile apps from 2014 till 2020.

One major finding was that the number of issues is substantially greater for Android than iOS due to the larger and more diverse range of supported devices, which adds a lot more complexity. There are presumably a number of factors that have contributed to this: i) Android uses Chromium as the default browser, whereas iOS uses the Safari browser by default; ii) Android has a large user base; iii) Chromium is rarely installed by iOS users, and iv) Android releases new versions of Chromium more frequently than iOS, possibly introducing new accessibility bugs given the reduced time to properly implement and evaluate new features.

Our findings suggest that the increasing number of accessibility users and reported accessibility issues indicate that developers need additional resources to address these issues and make their applications more accessible to people with disabilities.

RQ1 Summary
The increase in accessibility bug reports commenced in 2013, which may suggest that the community had grown more vital in reporting bugs or that users have increased over time. Most accessibility issues are related to Android because of the larger user base. Additionally, the rising trend of accessibility bug reports suggests that developers should allocate more resources to address these issues.

RQ$_2$: How does the time-to-fix of accessibility bug reports compares to the time-to-fix of other bug reports?

Motivation. Our main objective with this research question is to identify the amount of time required to solve accessibility-related bugs in comparison with other types of bugs. In addition, we inspect how accessibility bugs are classified and prioritized to have some insight into whether developers recognize that accessibility is an important feature. Understanding the developer's culture and rationale towards accessibility is important to come up with some explanation of why accessibility bugs may take a long time to be fixed or even not to be fixed.

Approach. The data was filtered and examined to discover the bug fixing time, which is the time taken to resolve a given bug. To calculate the fixing time, we defined the bug report time that was reported till the time that the bug was marked as "Fixed". Bug reports that were not reported with Fixed status were discarded from analysis, as this research question's objective is on bug reports that have been resolved. In this RQ, we want to compare accessibility and non-accessibility bug reports across platforms. When this analysis was achieved, the subset size evaluated for fixing time (i.e., the number of bugs reports analyzed) for accessibility-Android is 586 bug reports, for accessibility-iOS is 393 bug reports, while for non-accessibility in Android is 12,261 bug reports, and for non-accessibility in iOS is 3,112 bug reports. We further grouped the bug reports into four categories according to the number of days needed to fix them: 30 days or less, 31 days to 60 days, 61 days to 90 days, and More than 90 days. Next, we created a priority table to see which reports—accessibility or non-accessibility—were assigned the highest level of priority to be fixed. To achieve this, we divided the priorities into four levels, ranging from the most urgent (0) to the least urgent (3), as designated by Google's development team in the bug repository. Afterward, we utilized a Wilcoxon pairwise investigation on the data subsets to discover which platform combination holds the highest and lowest times to fix.

Results. Figure 5 shows a box plot for accessibility and non-accessibility issue resolution time in Android and iOS. The median number of days taken to fix issues was higher for accessibility-related issues than non-accessibility issues on both platforms. On average, it takes 52.5 median days to fix accessibility issues in Android and 83.0 median days in iOS (see Table 4). On the other hand, non-accessibility issues in iOS and Android were resolved in a relatively short period of time, i.e., 15.0 median days for Android and 17.0 median days for iOS. Another significant distinction is that fixing time in Android is considerably shorter than in iOS.

We also found that accessibility issues in Android and iOS are given less fixing priority than non-accessibility issues. For instance, only 0.34% accessibility issues are fixed for Android in less than 30 days, compared to 0.78% of non-accessibility issues (see Tables 5, 6, 7, and 8). It is worthwhile to point out that accessibility issues in Android are resolved with higher priority than those in iOS, with 38.46% (Table 5) of issues being resolved in the same period of time for Android as compared to 33.84% (Table 7) for iOS.

Finally, the Wilcoxon Rank-Sum hypothesis test was performed to determine the significance of the observed differences in the distribution of issue-fixing time. From the results in Table 9 we can see that the minimum value is for accessibility issues, while non-accessibility is also near 0.5, which is 0.218.

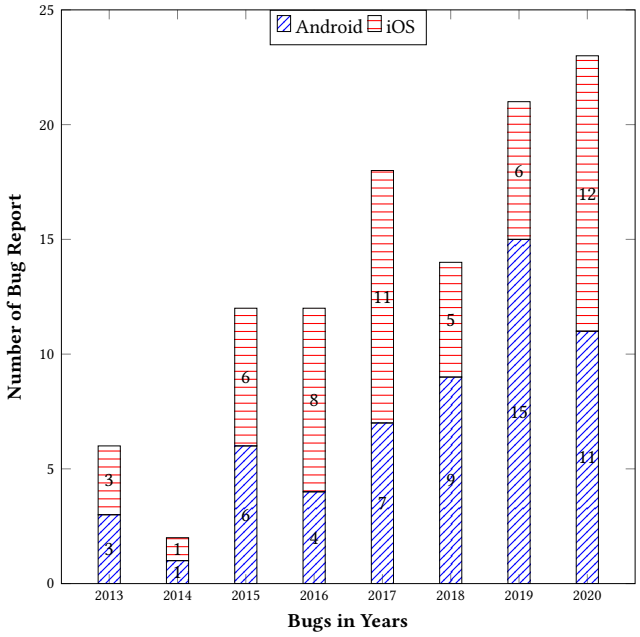

Figure 4: Yearly breakdown of the distribution of accessibility bug report for *open bug*.

Discussion. The results show that fixing time for accessibility bug reports is significantly longer than for non-accessibility bug reports, which can be because of the smaller volume of accessibility bugs reported and the large community involved in fixing the non-accessibility bugs. The developers seem to place a low priority on fixing accessibility bugs because of the small number of accessibility users. Our claim is supported by the cases presented in Tables 10, and 11. It is evident from Table 10 that accessibility users are not considered as "*real users*" by the developers, as stated by one of the project members:

> *Project Member. "Because it requires –enable-accessibility,* ***we reduce the severity****. Because it requires –force-renderer-accessibility, which* ***real users*** *don't actually use, we* ***lower severity again.****"*[2]

Another concerning matter is that developers prolong the fixing process by giving the report a lower priority because there are not many accessibility users. For instance, the Bug_ID:111656 was resolved on April 26, 2018, after being reported on Friday, January 27, 2012. Similar to this, Bug_ID:37721 took seven years for the developers to fix after being reported on March 9, 2010. Table 11 clarifies that the problem needs to be solved right away, but due to the small number of users who are actually affected, developers do not give accessibility bug reports a top priority. We also compared the length of relative bug-fixing time, which supports our claim that accessibility bug reports receive lower fixing priority than non-accessibility bug reports.

The Wilcoxon test results demonstrate that bugs in Android are resolved far more quickly than they are in iOS. It is interesting that bug-fixing time in Android for accessibility and non-accessibility

[2]https://bugs.chromium.org/p/chromium/issues/detail?id=111656

issues is shorter than in iOS, given that Android has a larger user base, more variants of supported devices, and considerably shorter release cycles. In any case, our data does not allow us to definitively explain why Android issues were resolved more quickly than iOS issues.

The much longer time to fix associated with the low priority frequently assigned to accessibility-related bugs is symptomatic of how accessibility is poorly handled in software development in general. Many recent studies [21, 39] have shown many reasons why accessibility is not considered during software development: lack of awareness and knowledge; lack of project or organization requirements related to accessibility; lack of time and resources; organizations prioritize functional features; lack of proper tools and more clear and practical standards. Not surprisingly, the same rationale seems to be applied when resources are assigned to fix accessibility-related bugs.

Figure 4 shows open accessibility bug reports for Android and iOS platforms, which have never been closed since it opened, spanning from 2013 to 2020. Notice that there are bugs that have remained open since 2013. Initially, the data suggest a potential underreporting or a nascent awareness of accessibility issues. As the years progress, both platforms exhibit an uptick in reports, with a notable divergence post-2017, where iOS bug reports substantially outpace those of Android. This uptrend, particularly the sustained open status of these reports, underscores a critical ongoing challenge in the software development lifecycle. The persistent open bugs, especially in accessibility, could reflect the intricacies involved in resolving such issues or possibly a lag in prioritization. This has significant implications, as unresolved accessibility bugs can severely hinder the user experience for individuals with disabilities, suggesting an area in need of targeted attention to foster inclusive technology use. The increasing trend also signals a heightened awareness and reporting of accessibility issues, yet it equally calls for more robust resolution mechanisms to ensure that such reports lead to timely and effective remedial actions.

Platform	Min.	Q1	Median	Mean	Q3	Max.
Android-A11y	0.0	13.0	52.5	255.3	309.0	2587.0
Android Non-A11y	0.00	4.00	15.00	81.78	65.00	2148.00
iOS-A11y	0.0	17.0	83.0	311.3	419.0	2587.0
iOS Non-A11y	0.00	4.00	17.00	94.97	75.00	2377.00

Table 4: Summary of fixing-time (in #days) for the two platforms.

RQ2 Summary

Fixing time for accessibility bug reports is higher than non-accessibility bug reports. Developers place a lower priority on accessibility bug reports due to the small user base. On the other hand, Android fixes accessibility-related reports faster than iOS.

RQ$_3$: What is the root cause of accessibility bug reports?

Motivation. Our main objective with this research question is to highlight the major factors that contribute to the accessibility issues in chromium applications. Although we recognize that these accessibility issues exist, we are still uncertain about the underlying causes. By identifying the root causes of accessibility bug reports, developers can make informed decisions during the development process and promote the adoption of accessibility principles.

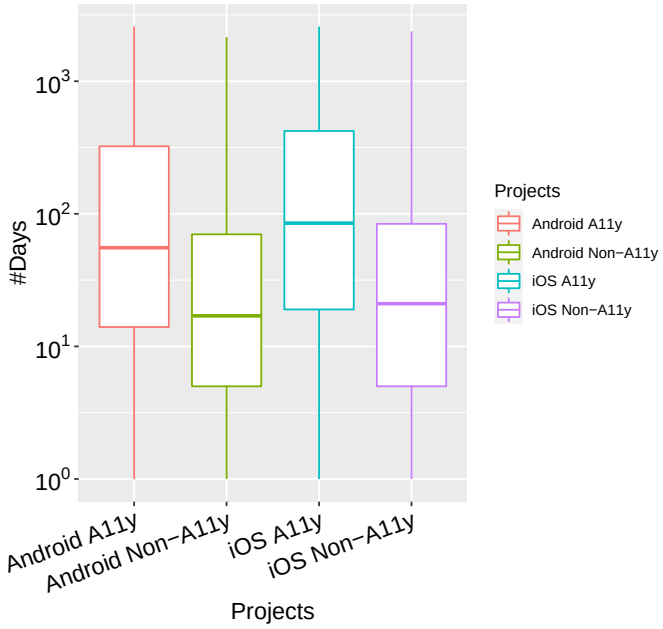

Figure 5: Box-plot of fixing-time (in #days) of accessibility (A11y) and non-accessibility (Non-A11y) issues for iOS and Android Platforms.

Approach. To identify the root cause of accessibility-related issues, we filtered and categorized the accessibility bug reports using an open-coding inspired process. The reports were distributed between two authors, who were instructed to read and assign multiple codes/keywords [15] related to the root causes of the accessibility issues in each bug report. The expertise of each author is aligned with the goal of our analysis. We refer to both authors as "coders" from this point forward, and they each labeled a total of 2,155 accessibility bug reports in three iterations.

In Iteration 1, the coders identified the keywords that pertain to the root cause of accessibility issues from the bug reports. The coders initiated an open discussion on building a taxonomy of the accessibility-related aspects based on the keywords. For instance, the keywords "*Aria elements, Aria Properties, and Aria relations*" correspond to the category "Aria". However, during this discussion, the coders identified an issue.

> *"Fred is hidden" (Bug ID: 87863)*
>
> *"M77 merge request: reverting previous merge which caused bug 1000874" (Bug ID: 1001024)*

The above bug reports share one commonality: they do not have any keywords that can be used to label the root cause of accessibility issues. However, the coders determined that these bug reports still pertain to accessibility-related issues. As a result, the coders categorized them as "Others". The coders agreed on 18 categories of the root causes of accessibility-related issues that are described in Table 13.

Table 5: Priority table for accessibility bug reports in Android.

Days	0 Most Urgent	1	2	3 Least Urgent	N/A	Total (%)	Total Count
30 days and less	0.34%	12.65%	21.71%	3.59%	0.17%	38.46%	225
31 days till 60 days	0%	3.93%	8.38%	0.68%	0%	12.99%	76
61 days till 90 days	0%	1.71%	3.93%	1.03%	0%	6.67%	39
More than 90 days	0.17%	5.98%	29.23%	6.32%	0.17%	41.88%	245
Total	0.51%	24.27%	63.25%	11.62%	0.34%	100.00%	585

Table 6: Priority table for non-accessibility bug reports in Android.

Days	0 Most Urgent	1	2	3 Least Urgent	N/A	Total (%)	Total Count
30 days and less	0.78%	26.72%	22.42%	13.24%	0.11%	63.27%	7758
31 days till 60 days	0.05%	3.61%	4.31%	2.77%	0.04%	10.79%	1323
61 days till 90 days	0.02%	1.61%	2.49%	1.65%	0%	5.77%	707
More than 90 days	0.03%	4.18%	9.47%	6.44%	0.05%	20.17%	2473
Total	0.88%	36.12%	38.69%	24.10%	0.20%	100.00%	12261

Table 7: Priority table for accessibility bug reports in iOS.

Days	0 Most Urgent	1	2	3 Least Urgent	N/A	Total (%)	Total Count
30 days and less	0%	9.16%	20.36%	4.07%	0.25%	33.84%	133
31 days till 60 days	0%	3.05%	8.40%	0.25%	0%	11.70%	46
61 days till 90 days	0%	0.76%	4.83%	1.02%	0%	6.62%	26
More than 90 days	0.25%	5.60%	34.86%	6.87%	0.25%	47.84%	188
Total	0.25%	18.58%	68.45%	12.21%	0.51%	100.00%	393

Table 8: Priority table for non-accessibility bug reports in iOS.

Days	0 Most Urgent	1	2	3 Least Urgent	N/A	Total (%)	Total Count
30 days and less	0.96%	21.53%	22.49%	15.68%	0.13%	60.80%	1892
31 days till 60 days	0.06%	3.31%	4.92%	3.02%	0%	11.31%	352
61 days till 90 days	0%	1.06%	2.96%	1.32%	0%	5.33%	166
More than 90 days	0.03%	3.63%	11.83%	7.07%	0%	22.56%	702
Total	1.06%	29.53%	42.19%	27.09%	0.13%	100.00%	3112

Table 9: P-values for Wilcoxon Rank-Sum tests.

Type	X vs. Y	Less	Greater	2-sided
Non Accessibility	Android vs. iOS	0.218	0.782	0.4359
Accessibility	Android vs. iOS	0.0232	0.9768	0.04639

In iteration 2, the coders identified and extracted keywords from the bug reports. In iteration 3, the coders classified the bug reports into the root causes based on the identified keywords. After classifying each bug report, the coders opened a discussion about the process of labeling them. The following is an example of one of the bug reports categorized as "Assistive Technology" since it includes the keyword "*talkback*".

> *"TalkBack doesn't read the text of the popup dialog when it appears." (Bug ID: 1114895)*

Following the guidelines of Aljedaani et al. [8], we validated the labeling procedures used by the coders to analyze and classify accessibility bug reports. We selected a 9% sample of the entire dataset of accessibility-related bug reports (190 bug reports), which satisfied a 95% confidence level and a confidence interval of 6. We then provided a random sample of 239 bug reports to the coders for labeling, which they had not previously seen. We used Cohen's Kappa Coefficient [26] to assess the inter-rater agreement between the two datasets, which resulted in an agreement level of 0.87, which is considered almost perfect agreement [34].

Results. As a result of the open-coding process, we identified 69 root causes of accessibility issues in Android and 96 in iOS from 2,157 bug reports. The root causes were then organized into 17 categories, which were based on the concepts and themes that emerged from the open coding process. From this point forward, we will be denoting the categories from the taxonomy in italic font.

Table 13 shows the number of accessibility issues pertaining to each category. The majority of the bug reports in both platforms were related to *Accessibility Improvements* in the Chromium application. This category refers to the recommendations made by the

Table 10: Example (Bug ID:111656) of user and project member comments on the reported accessibility issue.

Issue 111656: Security: Accessibility bad cast	
Reported by Anonymous	Fri, Jan 27, 2012 at 12:22 PM CST
Comment by Anonymous	Fri, Jan 27, 2012 at 1:11 PM CST
Labels: -SecSeverity-High SecSeverity-Low (was: NULL) Because it requires –enable-accessibility, we reduce the severity. **Because it requires –force-renderer-accessibility, which real users don't actually use, we lower severity again.**	
Comment by Anonymous	Fri, Jan 27, 2012 at 1:23 PM CST
To clarify, accessibility is enabled automatically for some users - Chrome auto-detects whether assistive technology is running and switches these flags on dynamically. The flags (–enable-accessibility and –force-renderer-accessibility) are used to force these code paths without installing third-party software or changing your operating system setup. **However, the fraction of users affected is low. I still think this is Pri-1 and not Pri-0**, I just wanted to clarify that these issues do affect real users, not just developers or testers.	
Comment by Anonymous	Fri, Jan 27, 2012 at 1:38 PM CST
Labels: -Pri-1 -SecSeverity-Low Pri-2 SecSeverity-Medium (was: NULL) Splitting the difference, SecSeverity-Medium. I'm making this a Pri-2 also. Vulnerabilities in the renderer are typically Pri-1 (rather than Pri-0) because of the sandbox. You're saying a small percentage of users have both of those flags automatically enabled, **so Medium/Pri-2 looks right to me.**	
Comment by Anonymous	Jan 29, 2012 at 6:46 PM CST
Please check with Justin, we have always treated such accessibility bugs with secseverity-low for renderer issues and medium for browser issues because very few users (think < 1%) run accessibility code or use assistive technology.	

Table 11: Example (Bug ID:37721) of user and project member comments on the reported accessibility issue.

Issue 37721: Skip links do not work when using screenreader.	
Reported by Anonymous	Mon, Mar 8, 2010 at 5:41 PM CST
Comment by Anonymous	Mon, Feb 27, 2012 at 4:21 AM CST
Big accessibility problem! Should be fixed ASAP!	
Comment by Anonymous	Thu, Mar 22, 2012 at 7:51 AM CDT
needs doing for mobile webkit as well.	
Comment by Anonymous	Jul 31, 2012 at 4:42 PM CDT
I experienced the same problem. **Needs a fix asap.**	
Comment by Anonymous	Fri, Oct 5, 2012 at 9:51 AM CDT
Workarounds are bad for you and the world, but thanks much for this temporary fix. :)	
Comment by Anonymous	Oct 11, 2012 at 9:25 AM CDT
Two years or more and this isn't fixed yet???	
Comment by Anonymous	Fri, Nov 30, 2012 at 11:25 AM CST
This is a terrible bug! Come on Google :D	
Comment by Anonymous	Fri, Nov 30, 2012 at 1:11 PM CST
Seriously.. And label your media controls. Come on. Patience is waning over here. :P	
Comment by Anonymous	Fri, Dec 7, 2012 at 8:30 AM CST
I was testing my accessibility requirements in Chrome and couldn't figure out why this wasn't working until now. **It's sad to see this has been ignored for so long.**	
Comment by Anonymous	Feb 23, 2013 at 1:56 AM CST
And still this has not been dealt with! A much needed fix for all webkit users especially since the browsers based on webkit together have now surpassed IE in internet usage. **Being unable to pass focus using fragmented hash links is very bad for accessibility.** Once a focused link is activated(enter pressed) it should pass keyboard focus to the element now in view!	
Comment by Anonymous	Mar 10, 2013 at 6:55 AM CDT
Labels: -Area-WebKit -Feature-Accessibility Cr-Content Cr-UI-Accessibility (was: NULL)	
Comment by Anonymous	Wed, Mar 13, 2013 at 6:26 AM CDT
Jeez, still not fixed after all this time - shame on you. ACCESSIBILITY MATTERS	

commentators to enhance accessibility for people with disabilities. Specifically, 331 (25.3%) of the 1308 bug reports on the Android platform and 220 (25.9%) of the 849 bug reports on the iOS platform were related to this were reported due to root causes relating to this category.

The second most highly reported category of accessibility bug reports in the Chromium application on both the Android and iOS platforms is *Design*. This category encompasses design-related issues that can have a significant impact on the usability, functionality, and overall effectiveness of the application. For example, design issues can make it difficult for users with disability to find and use features, or they can make the application difficult to navigate. The high number of design-related bug reports on both platforms suggests that there is a need for developers to improve the design of the Chromium application. This includes following accessibility guidelines to ensure that the application is accessible to users with disabilities. In the Android platform, 23.1% of the bug reports were related to *Design*, while in the iOS platform, 17.9% of the bugs were reported due to accessibility issues with *Design* of the application. These findings suggest that *Design*-related issues are a more significant problem in the Android platform than in the iOS platform.

Accessibility issues related to *Assistive Technologies* were the third most reported issues in the Chromium application. These issues amounted to 15.7% and 15.9% of the total bug reports for both Android and iOS platforms, respectively. *Assistive Technologies*, such as TalkBack and screen readers, are used by people with disabilities to access digital content. The significant number of accessibility issues related to *Assistive Technologies* indicates that developers need to improve the Chromium application's accessibility for users who rely on these technologies. This includes ensuring that the application is compatible with a wide range of *Assistive Technologies* and that the application provides clear and concise labels for all features. Furthermore, the higher number of reported bugs in this category for the iOS platform indicates that there are compatibility

issues between assistive technologies and iOS platforms, which suggests a critical need for improvement.

Focus-related issues were the fourth most common type of bug reported in Chromium applications, accounting for 10.7% of all bug reports in Android and 10.6% in iOS. These issues occur when the focus of an application is not handled correctly, making it difficult or impossible for people with disabilities to use the application. For example, people who use screen readers will not be able to interact with a button if it is not interactive.

Other categories for the reported root causes of accessibility issues in Chromium application include *API* (Android: 3.5%, iOS: 3.2%), *ARIA* (Android: 3.2%, iOS: 3.8%), *AX Properties* (Android: 2.5%, iOS: 3.4%), *Blink Accessibility* (Android: 1.2%, iOS: 1.6%), *Browser History* (Android: 0.5%, iOS: 0.8%), *Crash* (Android: 2.9%, iOS: 2.0%), *Forms* (Android: 2.9%, iOS: 3.7%), *Heap* (Android: 1.4%, iOS: 2.1%), *Links* (Android: 1.5%, iOS: 1.4%), *Notifications* (Android: 1.1%, iOS: 0.7%), *Platform* (Android: 1.6%, iOS: 1.6%), *Security* (Android: 0.9%, iOS: 0.7%), *Principles* (Android: 0.1%, iOS: 0.1%), and *Others* (Android: 0.9%, iOS: 4.0%).

Additionally, we provide the count of accessibility bug root causes for each category in Tables 14 and 14. It is evident that web accessibility principles and settings have the highest frequency of occurrence of accessibility issues, with a count of 98, followed by accessibility (count=96) in the category of *Accessibility Improvements* in the Android platform. On the contrary, the count for accessibility issues in the iOS platform is 72.

Similarly, the major root cause of accessibility issues related to *Assistive Technology* in Android platforms is UI elements and states (count=102), whereas in iOS, it is screen reading and feedback (count=81). In the case of *Design*, Accessibility Elements and Attributes cause the most accessibility issues, with a count of 94 in the Android platform. On the other hand, Web content and elements cause the most accessibility issues in the design of the Chromium application for the iOS platform, with a count of 65.

Discussion. The high number of *Accessibility Improvement* recommendations made by commentators suggests that there are a number of accessibility issues with the Chromium application on both platforms. For example,

> *"Use AccessibilityNodeInfo.setPaneTitle for web dialogs."* (Bug ID: 1048694)

The above-given bug report recommends the developers to use "AccessibilityNodeInfo.setPaneTitle" for web dialogues for improved accessibility of panes such as dialogue box or toolbar.

These issues could be due to a lack of compliance with accessibility standards, a lack of awareness of accessibility issues among developers, or a lack of resources available to developers to address accessibility issues. This also supports our analysis of RQ2, which found that developers do not prioritize accessibility issues, resulting in a high number of accessibility improvement recommendations. Moreover, the high percentage of these recommendations also suggests that there is a dire need for more accessible applications.

The fact that *Design* accounts for the second highest reported accessibility issue in the Chromium application reveals that developers need to focus on design-related elements as well. For instance, a user reported the following bug:

> *"Increment/Decrement of slider values do not work properly when percent change of a step is less than one."* (Bug ID: 475053)

This is important because it can make it difficult or impossible for people with disabilities to use the slider. For example, if a person with a visual impairment is using a screen reader, they may not be able to hear the difference between the different values of the slider. The issue with the slider is that it does not increment or decrement by the same amount each time. This can be confusing for users, especially those with disabilities. For example, a user with a visual impairment may not be able to tell the difference between a 1% change and a 2% change.

To fix this issue, the developers need to make sure that the slider increments and decrements by the same amount each time. They also need to provide clear and concise labels for the different values of the slider. This will help users, including those with disabilities, to understand how the slider works and to use it effectively. In addition to the issue with the slider, there are other design-related accessibility issues that need to be addressed in the Chromium application. For instance,

> *"Focused highlighting of ButtonCompat is difficult to see."* (Bug ID: 994365)

Bug_ID: 994365 presents an issue with visual accessibility since the focused highlighting of ButtonCompat is not visible enough.

> *"Site Settings button touch target too small"* (Bug ID: 968315)

Bug_ID: 968315 highlights the issue with the site setting button causing tactile accessibility issues. Our results also reveal a significant difference in the proportions of accessibility issues related to Android and iOS platforms. In Android, design-related issues account for 23.1%, while in iOS, they account for 17.9%, a difference of 5.2%. This suggests that there are more design-related accessibility issues in Android than in iOS. The design of an application has a major impact on its usability. Accessibility issues in the design of the Chromium application make it difficult and challenging for people with disabilities to use the application with ease. Developers should be proactive in addressing accessibility issues by making accessibility a priority early in the design process.

Assistive technologies are software or tools that help people with disabilities complete tasks that would otherwise be difficult or impossible. There are many Chromium-supported assistive technologies, such as TalkBack, ChromeVox, JAWS, VoiceOver, Dragon Naturally Speaking, and others. However, our analysis revealed that *Assistive Technology* is the third most reported issue in the Chromium application. This suggests that the assistive technologies that are used with the Chromium application have significant issues.

For example, skip links are used by users who rely on screen readers to navigate the Chromium application. Skip links are typically placed at the top and bottom of a page, and they allow users to quickly jump to specific sections of the page. However, if skip links are not working properly, people with disabilities may find it challenging to use the application. To illustrate this issue, a user reported the following bug:

> *"Skip links do not work when using screenreader."* (Bug ID: 37721)

In essence, users who rely on screen readers to navigate the Chromium application may not be able to use skip links to jump to specific sections of the page. This can make it difficult for them to use the application and access the information and services that are available on it. Another example of an assistive technology issue in the Chromium application is the buttons on the Manage Sync settings page. These buttons are not accessible to people who use Voiceover, a popular screen reader. Consequently, people who are blind or have low vision cannot use these buttons to interact with the Chromium application. To illustrate this issue, a user reported the following bug:

> *"Some buttons on Manage Sync settings page not read as buttons by Voiceover"* (Bug ID: 979696)

Bug_ID: 979696 reports that people who are blind or have low vision cannot use these buttons to interact with the Chromium application. This can make it difficult for them to use the application and manage their sync settings.

The remainder of the root causes for accessibility-related issues presented in Table 13 highlight the need for a variety of approaches to address the issues that exist. For example, bugs in the API layer may require changes to the Chromium codebase, while bugs in ARIA may be fixed by updating the ARIA implementation.

API layer refers to the set of application programming interfaces (APIs) that allow developers to interact with the Chromium application. Bugs in the API layer can prevent developers from creating accessible applications. For example, a bug in the Chromium application causes it to incorrectly handle fragment identifiers in data: URIs. This could cause problems for users who are trying to access specific sections of a web page that are embedded in data: URI.

> *"Incorrect handling of fragment identifiers in data: URIs"* (Bug ID: 123004)

Our analysis also revealed the root causes that hinder the accessibility of the Chromium application for users with disabilities. For example, one of the major root causes of accessibility issues in the Android platform related to *Assistive Technology* is talkback and voice interaction.

> *"TalkBack doesn't read the text of the popup dialog when it appears"* (Bug ID: 1114895)

On the other hand, in the iOS platform, the major root cause is screen reading and feedback. We have identified various root causes of accessibility issues, which can help developers address reports more easily.

> *"Keyboard gets dismissed on navigating through NTP with Voice over."* (Bug ID: 1146405)

Accessibility bugs in the Chromium application can severely hinder or prevent users with disabilities from using the application. For example, a blind user may not be able to use a button that is not labeled properly. Or, a user who has a motor impairment may not be able to use a form that is not properly designed for keyboard navigation. Therefore, it is crucial to address accessibility bugs in a timely manner. This includes fixing bugs in the API layer, updating the ARIA implementation, and educating developers about accessibility best practices. In essence, developers should address the root causes of accessibility issues in the Chromium application to render it more accessible to users with diverse abilities.

RQ3 Summary

The most common root causes of accessibility issues in Chromium are related to design, assistive technologies, focus, and non-compliance with accessibility standards. Additionally, the majority of commentators report suggestions and solutions for enhancing the accessibility of the application. Accessibility issues can make Chromium difficult to use for people with disabilities. Developers should address the root causes of these issues to make Chromium more accessible.

5 Takeaways

Our study provides an in-depth overview of the workflow for addressing accessibility-related bug reports on the Chromium application for Android and iOS platforms in comparison to traditional bug reports. Furthermore, we discussed the major root causes of these bug reports. In this section, we provide notable takeaways from our study.

Takeaway 1. As the number of accessibility-related bugs reported seems to be increasing over the years, either because users are more prone to report bugs or because accessibility violations are more prevalent, organizations should allocate adequate resources to ensure that their applications are accessible to everyone.

Takeaway 2. Users leverage different platforms (e.g., Twitter, app stores, GitHub issues, bug reports systems) to report accessibility bugs in the hope of improving mobile apps accessibility [20, 36, 45]. This is an indication that users are committed to participating in the design process by both reporting accessibility bugs and suggestions. Developers and organizations should seize the opportunity to engage users in devising more accessible products.

Takeaway 3. User feedback is essential to improve the quality of any digital product or service, either by reporting a bug to be fixed or by suggesting a new feature to be implemented [25, 46]. However, our study shows that accessibility-related bugs can receive less priority and take longer to be fixed, which is disappointing for users who expect their feedback will be useful and their demands will be addressed. Hence, it is crucial that developers and organizations start to consider accessibility as relevant as any other quality aspect to keep their users engaged and satisfied.

Takeaway 4. The prevalence of accessibility-related bugs in the Android platform compared to iOS suggests that developers of the Android platform should prioritize fixing accessibility issues. This might be a consequence of the fact that the number of users of Android devices has been increasing at a faster rate than iOS devices [54]. This requires Android developers to pay more attention to the accessibility of their applications. However, iOS developers should not ignore accessibility issues either.

Takeaway 5. Our research on the Chromium application for Android and iOS platforms has found that the most common root causes of accessibility issues are related to (i) the need for improvement in the application's accessibility features, (ii) accessibility design, and (iii) assistive technologies. Most of those issues could have been avoided if well-known standards and guidelines had been

Table 12: Summary of the accessibility identification and analysis in related work.

Study	Year	Purpose	Approach	Source of Info	Dataset	Platform Type
Ross et al. [50]	2017	Analysis accessibility barriers	Google a11y scanner	Google Play Store	100 apps	Android
Ross et al. [51]	2018	Analyze image-based button labeling	Epidemiology framework & Manual inspection	Rico repository	5,753 apps	Android
Eler et al. [33]	2019	Identify accessibility reviews	Manual analysis	User reviews	2,663 reviews	Android
Vendome et al. [55]	2019	Investigate accessibility posts	Manual analysis	StackOverflow	810	Android
Alshayban et al. [13]	2020	Investigate prevalence of accessibility issues	Statistical & Manual analysis	Google Play Store Survey	1,135 apps 61 participants	Android
Bi et al. [20]	2021	Investigate accessibility issues.	Manual Analysis	GitHub	11,820	Android, iOS, & Windows
Zhao et al. [64]	2023	Investigate interaction of accessibility issues with software quality attributes.	Manual Analysis	GitHub	4,572	Non-web application
This work	**2023**	**Analyze accessibility bug reports in open-source system**	**Statistical & Manual analysis**	**Bug reports**	**Android 1,308 iOS 849**	**Android & iOS**

followed. The fact that many accessibility violations were found on Chromium is in line with the results of some studies that suggest that, even though there are many resources and accessibility courses available, there is a lack of awareness and knowledge of developers concerning accessibility concepts and guidelines [21, 39]. In addition, the fact that accessibility is not a project or an organizational requirement jeopardizes adopting accessibility in the development process.

Takeaway 6. Our study manually analyzed and identified a set of keywords that are commonly used in accessibility-related bug reports, contributing to the definition of a taxonomy that can help identify accessibility bugs in other software systems. As manually analyzing bug descriptions can be labor-intensive and error-prone, we recommend that a system be developed based on these keywords to automate the detection of accessibility-related bug reports.

6 Related Work

In the application creation and maintenance of large-scale applications, bug report databases have been a vital resource. They offer the app developers helpful information and allow end-users to inform application developers of challenges encountered by users through the use of the app. Many researchers mostly adopt bug repositories because of their importance. In this section, we describe three facets of related studies, demonstrating the variations between our analysis and the related studies.

6.1 Previous Empirical Studies in Software Accessibility

Previous studies have conducted empirical investigations on different aspects of mobile accessibility. For instance, Serra et al. [53] performed a manual evaluation of Brazilian government apps on both iOS and Android platforms using WCAG 2.0 guidelines [56]. They found that the issues identified in accessibility are extensively discovered in the evaluated applications. Another study analyzed the interaction of blind users by VoiceOver on iOS devices [40]. Their study involved 55 blind users in answering the survey to determine user satisfaction, and the authors found some usability issues. Milne et al. [42] studied the accessibility for blind users using heath sensors for nine commercial applications in the iOS platform, while Walker et al. [59] developed a weather app with a universal design for both platforms to evaluate the accessibility due to the existing apps are not universally accessible.

In more recent work, Vendome et al. [55] analyzed 1,442 StackOverflow discussions (questions & answers) to identify posts related to Android accessibility. The goal of their study is to understand the developers' issues and accessibility practices. Their approach used keyword-matching techniques to identify Android issues, and their results were manually validated. The authors' findings showed that 810 posts were related to Accessibility in Android. In another study, Alshayban et al. [13] performed an empirical study to understand the accessibility issues on Android apps. Their study examined 1,000 Android apps and 61 developers' responses survey-based. Their results showed that 48.53% of developers lack awareness of accessibility aspects. Zhang et al. [63] attempted to mitigate the color-related accessibility issues in Android platforms. Zhao et al. [64] investigated the interaction between accessibility issues and other quality factors that affect software performance. The researchers collected 4,572 accessibility issues from GitHub and manually assessed them to determine the violated standards, causes, and solutions. The study revealed that usability and functionality are the quality attributes that are most frequently intertwined with accessibility.

Other studies have also previously looked into particular problems related to mobile accessibility, such as user reviews [3, 7, 10, 32, 33], Code Elements [27], accessibility barriers [50], missing labels [23], alternative image labels [42, 53], and alternative text labels [51]. However, our study is different from any prior studies in both purpose and method. Table 12 summarizes prior studies conducted on accessibility-related issues reported by users in different platforms. Our purpose is to identify and analyze accessibility bug reports in open-source software across platforms instead of focusing on one platform, such as Android, to compare frequency, fixing time, and types of accessibility bug reports.

6.2 Bug Repository and Qualitative Analysis

Several prior studies have been conducted on the qualitative analysis of bug repositories, such as ranking developer [11, 43, 60], bug evolution [4, 16], survey [17, 18, 62], and ranking faulty source files [44, 52].

Table 13: Taxonomy & count of accessibility bug reports in Android and iOS platforms.

Category	Android Count	iOS Count	Description	Sample Bug Report
Accessibility Improvements	331	220	Recommendations and suggestions to ensure the improvement in accessibility features of software or applications, making them more usable and inclusive for individuals with disabilities.	*"Implement native accessibility support for out-of-process iframes." (Bug ID:368298)*
Assistive Technology	206	135	Bug reports that pertain to the functionality, compatibility, and effectiveness of the assistive tools (e.g. voiceover, talkback, screen reader, etc.) developed to enhance accessibility.	*"Voiceover should alert suggestions available when autofill appears" (Bug ID: 532721)*
API	46	28	These bug reports correspond to test cases or functions within a testing framework or tool.	*"UKMTestCase.testHistoryDelete is flaky on iOS simulator." (Bug ID: 866598)*
Aria	43	33	ARIA aids assistive technologies in accessing dynamic and interactive web content.	*"HTML table with ARIA roles appears empty in the accessibility tree." (Bug ID: 372708)*
AX Properties	34	29	Set of attributes defined by Chromium that pertain to the accessibility of web content via assistive technology and testing.	*"Media Router WebUI: false positives for AX_FOCUS_01 audit" (Bug ID: 514795)*
Blink Accessibility	16	14	Features and tools that are intended to improve the accessibility of web content for individuals with impairments.	*"Blink should be firing AXEventBlur." (Bug ID: 534619)*
Browser History	7	7	Accessibility issues related to browser history and settings.	*"history: treat history entry list as single control to shorten tab order" (Bug ID: 385328)*
Crash	38	17	Crash bugs due to accessibility feature usage.	*"Chrome accessibility crash." (Bug ID: 29154)*
Design	303	152	Design-related accessibility issues, such as those related to user interface design and system architecture, can significantly impact the usability and functionality of an application.	*"Contrast too low on bookmark/history page empty view." (Bug ID: 967817)*
Focus	141	90	Usability issues that occur when elements in an application cannot be easily accessed or navigated.	*"Talkback focus rectangle positioned incorrectly." (Bug ID: 978512)*
Forms	38	32	Usability issues that hinder the accessibility of forms by people with impairments such as unclear labels and error messages, etc.	*"Label names when they embed a control are not correct." (Bug ID: 1121637)*
Heap	19	18	Memory leaks that cause unpredictable behavior, such as crashes, data corruption, and performance degradation.	*"Heap-use-after-free in WebCore::AXObjectCache::postNotification" (Bug ID: 127371)*
Links	20	12	Link navigation issues that arise due to the absence of redirect warnings, inconsistent and broken links.window.close()"	*"Open link in new tab" should not allow" (Bug ID: 170757)*
Notifications	15	6	Inaccessible notifications due to poor visibility or inaudibility for people with impairments.	*"Audit all bubbles to determine if they should be alert dialogs" (Bug ID: 474622)*
Platform	21	14	Accessibility-related issues that are concerned with the operating system.	*"Live regions support flakey" (Bug ID: 535250)*
Security	13	6	Issues that occur due to inaccessibility or poor implementation of security features in the application.	*"Security indicators should be accessible." (Bug ID: 447191)*
Principles	2	1	Issues that are related to violations of accessibility principles.	*"Chrome is 3x slower than Safari in a benchmark"*
Others	13	34	Undefined	*"Fred is hidden" (Bug ID: 87863)*
Grand Total	1308	849		

Table 14: Root causes related to each category of accessibility bug report.

Category	Root Causes (Android)	Count (Android)	Root Causes (iOS)	Count (iOS)
Accessibility Improvements	Accessibility	96	Accessibility API & Events	14
	User Interface Elements & Interactions	8	Accessibility Configuration & Flags	6
	Accessibility & Browser Behavior	5	Accessibility Elements & Attributes	46
	Accessibility & Document Structure	5	Accessibility in Multimedia	3
	Accessibility & Media	3	Accessibility Issues	72
	Accessibility & UI Components	5	Accessibility Performance	3
	Accessibility & User Preferences	10	Accessibility Tags & Elements	21
	Accessibility Elements & Nodes	35	Accessibility Testing & Debugging	21
	Accessibility Events	9	Accessibility Testing Framework & Utilities	1
	Accessibility in Multilingual Contexts	1	Accessibility Tools & Debugging	5
	Accessibility in Third-party Libraries	1	Automated Accessibility Testing	9
	Accessibility Issues & Problem Solving	10	Browser Accessibility & Elements	12
	Audio & Speech Recognition	6	Interactions with Web UI	2
	HTML & Document Accessibility	5	Language Accessibility	3
	Memory & Performance Issues	4	User Interface & Navigation Accessibility	2
	User Interface Elements & Interactions	8		
	Web Accessibility & Testing	30		
	Web Accessibility Principles & Settings	98		
API	Automated Testing	15	Accessibility Testing APIs	6
	Debugging & Logging	8	Context & Selection	5
	KeyboardInaccessibleWidget	1	Layout Testing	5
	Layout & Rendering	9	namespace	1
	Testing Framework & Utilities	13	Native Accessibility	1
			Notification Testing	1
			PDF Accessibility	2
			Rendering & Views	3
			Tab Management	2
			Toolbar & View Testing	1
			Web Worker Testing	1
Aria	Accessibility & ARIA Relations	3	Aria elements	12
	ARIA Descriptions & Labels	17	Aria Properties	17
	ARIA Roles & Attributes	19	Aria relations	4
			Testing & Verification	4
Assistive Technology	Accessibility Extensions & Features	10	Accessibility Announcements	4
	Accessibility Settings & Tools	4	Accessibility Tree & Navigation	21
	Screen Readers & Principles	18	Audio & Speech Recognition	7
	Talkback & Voice Interaction	63	ChromeVox	10
	Testing & Debugging	9	DumpAccessibilityTreeTest	2
	UI Elements & States	102	Performance & Benchmarks	2
			Screen Reading & Feedback	81
			Tools	6
			Touch & Interaction	2
AX properties	Accessibility APIs & Integration	5	Accessibility Events & Interaction	7
	Accessibility Events & Rules	4	Accessibility Integration & Compatibility	3
	Accessibility Tree (AX Tree)	15	AX Attributes	15
	Accessibility UI Elements & Objects	10	Text & Character Accessibility	4
Blink Accessibility	Blink Accessibility	16	Blink Accessibility	14
Browser History	Browser history	7	Browser history	7
Crash	Crash Causes & Scenarios	38	Accessibility crash	1
			Accessibility mode, crash	1
			AccessibilityRenderObject, Crash	1
			app freeze	1
			aria-owns parent, crash	1
			Chrome crash	3
			Crash, voice over	1
			Debug mode, Crash	1
			Gmail crash, BrowserAccessibilityMAnager	1
			HWNDViewContainer	1
			Inspect32, Accessibility crash	1
			SVG navigation crash	1
			tab crash	1
			v8_shell, crash	1
			Voice over, crash	1

Table 14: Root causes related to each category of accessibility bug report. (Cont.)

Category	Root Causes (Android)	Count (Android)	Root Causes (iOS)	Count (iOS)
Design	Accessibility & Design	48	Accessibility &Usability Features	25
	Accessibility Elements & Attributes	94	Accessibility Tools & Developer Resources	7
	Animations & Images	5	Color & Design	11
	Browser & Page Behavior	24	Text & Typography	25
	Keyboard & Input	11	User Interaction & Behavior	18
	Tables & Structured Content	15	Web Content & Elements	65
	Text & Typography	35		
	UI & User Interaction	32		
	Web Content & Elements	39		
Focus	Auto-refresh & Dynamic Content	9	Accessibility Highlights	2
	Focus & Accessibility Focus	99	Accessibility of User Interface Elements	20
	Keyboard Focus & Navigation	28	Exposing font/language	1
	Omnibox & Dropdown Focus	5	Focus Handling & Navigation	45
			Keyboard Accessibility & Interaction	18
			User Input & Interaction	4
Forms	Checkbox & Toggle	2	Attributes	2
	Default Values & Messages	10	autocomplete	1
	Forms & Form Elements	19	Data Input & Formatting	4
	Input & Input Types	7	Error Handling	3
			Form Interaction	5
			HTML Elements & Attributes	15
			User Interface Elements	2
Heap	Heap	19	BoundsForCharacter	1
			firstAbstractInlineTextBox	0
			Heap	8
			Heap-use-after-free	5
			notificationPostTimerFired	1
			remoteSVGRootElement	2
			visiblePositionForIndex	1
Links	Navigation & Links	20	DCHECK	1
			Link	3
			Navigation	8
Notifications	Notifications	15	Notifications	6
Platforms	Chrome & Platform	4	Operating Systems	7
	Configuration & Compression	2	Web Accessibility	4
	Interstitials & Layout	4	Web Development & Debugging	3
			Lint & Debugging	2
			Live regions	5
			Message Threads & OS	3
			select multiple	1
Principles	Accessibility	1	Accessibility	1
			Talkback	1
Security	Security	13	Security	6

Zhou et al. [65] performed a qualitative analysis on 88 open-source projects on desktop and mobile platforms. Their study goal is to understand the similarities and differences between desktop and mobile platforms in terms of bug-fixing and its process. In another study, Bhattacharya et al. [19] conducted an empirical study on Android bug reports. Their study was performed on 24 Android apps for several categories, such as communication, education, and heath-fitness. The authors used various metrics to analyze bug fixing time, bug-fixing process, bug priority, bug status, and bug category. Maji et al. [41] analyzed the Android platform on 233 bug reports for one year, 2008-2009, to categorize bug reports into types based on the code modification. Banerjee et al. [14] compared bug reports of Eclipse and Mozilla to identify both projects' similarities and differences. They analyzed bug report frequency, bug report types, bug duplication, and user behavior.

In a study similar to ours, Aljedaani et al. [9] conducted an empirical investigation focusing on iOS and Android platforms within the Mozilla and Chromium ecosystems. The primary objective of their research was to scrutinize the frequency, resolution time, and categorization of software bugs. In their methodology, they applied the Wilcoxon Rank-Sum test to ascertain the resolution time, revealing noteworthy findings. Specifically, they observed that, on the Mozilla platform, the median time to resolve a reported bug was 12 days for Android and 8 days for iOS. Meanwhile, on the Chromium platform, they found that bug reports took an average of 21 days to resolve on Android and 8 days on iOS, aligning closely with the Mozilla results. Another study [5] comparing the resolution times of security-related bug reports in Chromium projects noted faster resolution on Linux (median 10 days) than on Windows (median 14 days). A different study [6] performed an analysis of Eclipse bug reports logged on the Bugzilla platform. Their investigation revolved around the prioritization of bug reports and their impact on the average resolution time. Notably, their findings indicated that a minority of bug reports received an urgent priority rating, whereas

the majority of reports fell within the high to medium priority spectrum. Compared to the resolution time of accessibility-related bug reports, security and other "traditional bug reports" typically received higher priority. This differentiation in priority allocation is noteworthy, as it underscores the perception that accessibility concerns may, at times, be assigned a lower priority. This can be attributed to a common misconception that accessibility issues do not significantly impact a large number of users.

Our work is similar in their approaches but not in their purposes. We applied similar metrics as [9] to analyze bug frequency and bug fixing time. All previous studies [9, 14, 19, 41, 65] applied the analysis on general bug reports or comparing mobile platforms. However, we perform a comparison of accessibility and non-accessibility-related bug reports. To the best of our knowledge, this is the first empirical study that analyzes accessibility bug reports in bug frequency, bug-fixing time, and bug types for Chromium Android and iOS platforms.

7 Threats and Validity

Internal Validity. The dataset was gathered from the Chromium database. For RQ2, we calculated fixing time for only bug reports marked as "Fixed" based on the time we collected the dataset. In the future, this could inadvertently be skewed if any of these bug reports could be re-opened at a later stage.

External Validity. In this study, the analyses were done on web browser domains, specifically on the Android and iOS of Chromium. The selection was made due to the ease of accessibility to the database and public availability. However, there are other projects that provided a cross-platform application, such as Firefox, were not considered in this analysis because Chromium provided a bug label for each bug report, for this, we have the confidence that our accessibility-related bug reports are referred to as accessibility issue, rather than using keywords to match the accessibility-related bug reports.

8 Conclusion and Future Work

In this study, we investigated 39,160 bug reports on the Chromium application for Android and iOS platforms, which were gathered from the Monorail bug tracker repository. We analyzed the accessibility-related bug reports to identify trends in the reporting of these bugs across both platforms. Our study found that the number of accessibility-related bug reports has been increasing, which may be due to the increasing number of accessibility users and applications. It is worth noting that the Android platform has a higher number of accessibility issues than iOS. This may be due to the larger user base of Android, as well as the wider variety of devices and applications available on the platform. We also found that accessibility-related bugs are often given lower priority than non-accessibility-related bugs, as developers may perceive accessibility issues as having less impact on a large user base.

Through open coding and statistical analysis, we found that more than 25% accessibility issues in the Chromium application are related to the need for improvement in accessibility features. This is followed by design-related issues (23.4% on Android, 18.1% on iOS) and assistive technology concerns (16.2% on both platforms). These findings provide an initial exploration of the root causes of accessibility issues in the Chromium application. Our findings also highlight the opportunity for developers to overcome accessibility issues by addressing the main root causes, such as ARIA, security, AX properties, and other accessibility guidelines.

References

[1] IBM Accessibility. 2017. IBM Accessibility Checklist for 7.0. https://www.ibm.com/able/guidelines/ci162/accessibility_checklist.html.

[2] IBM Accessibility. 2021. Design, build, and test like an accessibility expert. https://www.ibm.com/able/.

[3] Wajdi Aljedaani, Mohammed Alkahtani, Stephanie Ludi, Mohamed Wiem Mkaouer, Marcelo M Eler, Marouane Kessentini, and Ali Ouni. 2023. The state of accessibility in blackboard: Survey and user reviews case study. In *Proceedings of the 20th International Web for All Conference*. 84–95.

[4] Wajdi Aljedaani and Yasir Javed. 2018. Bug Reports Evolution in Open Source Systems. In *5th International Symposium on Data Mining Applications*. Springer, 63–73.

[5] Wajdi Aljedaani, Yasir Javed, and Mamdouh Alenezi. 2020. LDA categorization of security bug reports in chromium projects. In *Proceedings of the 2020 European symposium on software engineering*. 154–161.

[6] Wajdi Aljedaani, Yasir Javed, and Mamdouh Alenezi. 2020. Open source systems bug reports: Meta-analysis. In *Proceedings of the 2020 The 3rd International Conference on Big Data and Education*. 43–49.

[7] Wajdi Aljedaani, Mohamed Wiem Mkaouer, Stephanie Ludi, and Yasir Javed. 2022. Automatic classification of accessibility user reviews in android apps. In *2022 7th international conference on data science and machine learning applications (CDMA)*. IEEE, 133–138.

[8] Wajdi Aljedaani, Mohamed Wiem Mkaouer, Stephanie Ludi, Ali Ouni, and Ilyes Jenhani. 2022. On the identification of accessibility bug reports in open source systems. In *Proceedings of the 19th international web for all conference*. 1–11.

[9] Wajdi Aljedaani, Meiyappan Nagappan, Bram Adams, and Michael Godfrey. 2019. A comparison of bugs across the iOS and Android platforms of two open source cross platform browser apps. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 76–86.

[10] Wajdi Aljedaani, Furqan Rustam, Stephanie Ludi, Ali Ouni, and Mohamed Wiem Mkaouer. 2021. Learning sentiment analysis for accessibility user reviews. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 239–246.

[11] Bader Alkhazi, Andrew DiStasi, Wajdi Aljedaani, Hussein Alrubaye, Xin Ye, and Mohamed Wiem Mkaouer. 2020. Learning to rank developers for bug report assignment. *Applied Soft Computing* 95 (2020), 106667.

[12] Eman Abdullah AlOmar, Wajdi Aljedaani, Murtaza Tamjeed, Mohamed Wiem Mkaouer, and Yasmine N El-Glaly. 2021. Finding the needle in a haystack: On the automatic identification of accessibility user reviews. In *Proceedings of the 2021 CHI conference on human factors in computing systems*. 1–15.

[13] Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility issues in Android apps: state of affairs, sentiments, and ways forward. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1323–1334.

[14] Sean Banerjee, Jordan Helmick, Zahid Syed, and Bojan Cukic. 2015. 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*. IEEE, 263–270.

[15] BBC. 2020. The BBC Standards and Guidelines for Mobile Accessibility. https://www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile.

[16] Mario Luca Bernardi, Gerardo Canfora, Giuseppe A Di Lucca, Massimiliano Di Penta, and Damiano Distante. 2012. Do developers introduce bugs when they do not communicate? the case of eclipse and mozilla. In *2012 16th European Conference on Software Maintenance and Reengineering*. IEEE, 139–148.

[17] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiß, Rahul Premraj, and Thomas Zimmermann. 2007. Quality of bug reports in eclipse. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*. 21–25.

[18] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 308–318.

[19] Pamela Bhattacharya, Liudmila Ulanova, Iulian Neamtiu, and Sai Charan Koduru. 2013. An empirical analysis of bug reports and bug fixing in open source android apps. In *2013 17th European Conference on Software Maintenance and Reengineering*. IEEE, 133–143.

[20] Tingting Bi, Xin Xia, David Lo, and Aldeida Aleti. 2021. A First Look at Accessibility Issues in Popular GitHub Projects. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 390–401.

[21] Tingting Bi, Xin Xia, David Lo, John Grundy, Thomas Zimmermann, and Denae Ford. 2022. Accessibility in Software Practice: A Practitioner's Perspective. *ACM Trans. Softw. Eng. Methodol.* 31, 4, Article 66 (jul 2022), 26 pages. https://doi.org/10.1145/3503508

[22] United States Access Board. 2017. Information and Communication Technology (ICT) Final Standards and Guidelines. https://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-ict-refresh/final-rule.

[23] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhut, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 322–334.

[24] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2022. Accessible or Not? An Empirical Investigation of Android App Accessibility. *IEEE Transactions on Software Engineering* 48, 10 (2022), 3954–3968. https://doi.org/10.1109/TSE.2021.3108162

[25] Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald C Gall. 2017. Analyzing reviews and code of mobile apps for better release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 91–102.

[26] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[27] Henrique Neves da Silva, Andre Takeshi Endo, Marcelo Medeiros Eler, Silvia Regina Vergilio, and Vinicius HS Durelli. 2020. On the Relation between Code Elements and Accessibility Issues in Android Apps. In *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*. 40–49.

[28] Android Developers. 2021. Accessibility Developer Checklist. https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/ui/accessibility/checklist.html.

[29] Android Developers. 2021. Build more accessible apps. https://developer.android.com/guide/topics/ui/accessibility.

[30] Apple Developers. 2021. iOS Human Interface Guidelines for Accessibility. https://developer.apple.com/design/human-interface-guidelines/.

[31] Google Developers. 2021. Accessibility Developer Checklist. https://developers.google.com/web/fundamentals/accessibility.

[32] Paulo Sérgio Henrique Dos Santos, Alberto Dumont Alves Oliveira, Thais Bonjorni Nobre De Jesus, Wajdi Aljedaani, and Marcelo Medeiros Eler. 2023. Evolution may come with a price: analyzing user reviews to understand the impact of updates on mobile apps accessibility. In *Proceedings of the XXII Brazilian Symposium on Human Factors in Computing Systems*. 1–11.

[33] Marcelo Medeiros Eler, Leandro Orlandin, and Alberto Dumont Alves Oliveira. 2019. Do Android app users care about accessibility? an analysis of user reviews on the Google play store. In *Proceedings of the 18th Brazilian Symposium on Human Factors in Computing Systems*. 1–11.

[34] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. john wiley & sons.

[35] Cassio Andrade Furukawa, Michele dos Santos Soares, Maria Istela Cagnin, and Débora Maria Barroso Paiva. 2022. Support for Accessible Software Coding: Results of a Rapid Literature Review. *CLEI electronic journal* 25, 3 (2022), 1–1.

[36] Syed Fatiul Huq, Abdulaziz Alshayban, Ziyao He, and Sam Malek. 2023. #A11yDev: Understanding Contemporary Software Accessibility Practices from Twitter Conversations. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (2023). https://api.semanticscholar.org/CorpusID:258217085

[37] iOS Chrome Releases. 2021. Google Chrome - The Fast and Secure Web Browser. https://www.appannie.com/apps/ios/app/chrome-web-browser-by-google/details/.

[38] Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. 2012. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *2012 9th IEEE Working Conference on Mining Software Repositories (MSR)*. IEEE, 179–188.

[39] Manoel Victor Rodrigues Leite, Lilian Passos Scatalon, André Pimenta Freire, and Marcelo Medeiros Eler. 2021. Accessibility in the mobile development industry in Brazil: Awareness, knowledge, adoption, motivations and barriers. *Journal of Systems and Software* 177 (2021), 110942. https://doi.org/10.1016/j.jss.2021.110942

[40] Barbara Leporini, Maria Claudia Buzzi, and Marina Buzzi. 2012. Interacting with mobile devices via VoiceOver: usability and accessibility issues. In *Proceedings of the 24th Australian Computer-Human Interaction Conference*. 339–348.

[41] Amiya Kumar Maji, Kangli Hao, Salmin Sultana, and Saurabh Bagchi. 2010. Characterizing failures in mobile oses: A case study with android and symbian. In *2010 IEEE 21st International Symposium on Software Reliability Engineering*. IEEE, 249–258.

[42] Lauren R Milne, Cynthia L Bennett, and Richard E Ladner. 2014. The accessibility of mobile health sensors for blind users. (2014).

[43] Hoda Naguib, Nitesh Narayan, Bernd Brügge, and Dina Helal. 2013. Bug report assignee recommendation using activity profiles. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 22–30.

[44] Anh Tuan Nguyen, Tung Thanh Nguyen, Jafar Al-Kofahi, Hung Viet Nguyen, and Tien N Nguyen. 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 263–272.

[45] Alberto Dumont Alves Oliveira, Paulo Sérgio Henrique Dos Santos, Wilson Estécio Marcílio Júnior, Wajdi M Aljedaani, Danilo Medeiros Eler, and Marcelo Medeiros Eler. 2023. Analyzing Accessibility Reviews Associated with Visual Disabilities or Eye Conditions. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 37, 14 pages. https://doi.org/10.1145/3544548.3581315

[46] S. Panichella, A. Di Sorbo, E. Guzman, C. A. Visaggio, G. Canfora, and H. C. Gall. 2015. How can i improve my app? Classifying user reviews for software maintenance and evolution. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 281–290. https://doi.org/10.1109/ICSM.2015.7332474

[47] Android Chrome Releases. 2021. Google Chrome: Fast & Secure. https://www.appannie.com/apps/google-play/app/20600000234348/details/.

[48] Chrome Releases. 2021. Release updates from the Chrome team. https://chromereleases.googleblog.com/search/label/Chrome%20for%20Android.

[49] Willem Rens. 2017. Browser forensics: adblocker extensions.

[50] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2017. Epidemiology as a framework for large-scale mobile application accessibility assessment. In *Proceedings of the 19th international ACM SIGACCESS conference on computers and accessibility*. 2–11.

[51] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O Wobbrock. 2018. Examining image-based button labeling for accessibility in Android apps through large-scale analysis. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*. 119–130.

[52] Nasir Safdari, Hussein Alrubaye, Wajdi Aljedaani, Bladimir Baez Baez, Andrew DiStasi, and Mohamed Wiem Mkaouer. 2019. Learning to rank faulty source files for dependent bug reports. In *Big Data: Learning, Analytics, and Applications*, Vol. 10989. International Society for Optics and Photonics, 109890B.

[53] Leandro Coelho Serra, Lucas Pedroso Carvalho, Lucas Pereira Ferreira, Jorge Belimar Silva Vaz, and André Pimenta Freire. 2015. Accessibility evaluation of e-government mobile applications in Brazil. *Procedia Computer Science* 67 (2015), 348–357.

[54] Shubham Shakya and Mayank Dave. 2022. Analysis, detection, and classification of android malware using system calls. *arXiv preprint arXiv:2208.06130* (2022).

[55] Christopher Vendome, Diana Solano, Santiago Linán, and Mario Linares-Vásquez. 2019. Can everyone use my app? an empirical study on accessibility in android apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 41–52.

[56] World Wide Web Consortium (W3C). 2008. Web Content Accessibility Guidelines (WCAG) 2.0. https://www.w3.org/TR/WCAG20/.

[57] World Wide Web Consortium (W3C). 2021. Web Content Accessibility Guidelines (WCAG) 2.2. https://www.w3.org/TR/WCAG22/.

[58] Web Accessibility Initiative (WAI). 2015. Mobile Accessibility: HowWCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile. http://www.w3.org/TR/mobile-accessibility-mapping.

[59] Bruce N Walker, Brianna J Tomlinson, and Jonathan H Schuett. 2017. Universal design of mobile apps: Making weather information accessible. In *International Conference on Universal Access in Human-Computer Interaction*. Springer, 113–122.

[60] Jifeng Xuan, He Jiang, Zhilei Ren, and Weiqin Zou. 2012. Developer prioritization in bug repositories. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 25–35.

[61] Shunguo Yan and PG Ramachandran. 2019. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing (TACCESS)* 12, 1 (2019), 1–31.

[62] Jie Zhang, Xiaoyin Wang, Dan Hao, Bing Xie, Lu Zhang, and Hong Mei. 2015. A survey on bug-report analysis. *Science China Information Sciences* 58, 2 (2015), 1–24.

[63] Yuxin Zhang, Sen Chen, Lingling Fan, Chunyang Chen, and Xiaohong Li. 2023. Automated and Context-Aware Repair of Color-Related Accessibility Issues for Android Apps. *arXiv preprint arXiv:2308.09029* (2023).

[64] Yaxin Zhao, Lina Gong, Wenhua Yang, and Yu Zhou. 2023. How accessibility affect other quality attributes of software? A case study of GitHub. *Science of Computer Programming* (2023), 103027.

[65] Bo Zhou, Iulian Neamtiu, and Rajiv Gupta. 2015. A cross-platform analysis of bugs and bug-fixing in open source projects: Desktop vs. android vs. ios. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. 1–10.