Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward

Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek
University of California, Irvine, USA
{aalshayb,iftekha,malek}@uci.edu

ABSTRACT

Mobile apps are an integral component of our daily life. Ability to use mobile apps is important for everyone, but arguably even more so for approximately 15% of the world population with disabilities. This paper presents the results of a large-scale empirical study aimed at understanding accessibility of Android apps from three complementary perspectives. First, we analyze the prevalence of accessibility issues in over 1,000 Android apps. We find that almost all apps are riddled with accessibility issues, hindering their use by disabled people. We then investigate the developer sentiments through a survey aimed at understanding the root causes of so many accessibility issues. We find that in large part developers are unaware of accessibility design principles and analysis tools, and the organizations in which they are employed do not place a premium on accessibility. We finally investigate user ratings and comments on app stores. We find that due to the disproportionately small number of users with disabilities, user ratings and app popularity are not indicative of the extent of accessibility issues in apps. We conclude the paper with several observations that form the foundation for future research and development.

ACM Reference Format:
Abdulaziz Alshayban, Iftekhar Ahmed, and Sam Malek. 2020. Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3377811.3380392

1 INTRODUCTION

Mobile applications (apps) play an important role in the daily life of billions of people around the world, from personal banking to communication, to transportation, and more. Ability to access these necessary services with ease is important for everyone, especially for approximately 15% of the world population with disabilities [53]. As app usage has steadily increased over the years among the disabled people, so has their reliance on the accessibility features. In a survey conducted in 2017, it was found that 90.9% of visually impaired respondents used screen readers on a smartphone [51], which is substantially higher than prior years.

ICSE '20, May 23–29, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7121-6/20/05…$15.00
https://doi.org/10.1145/3377811.3380392

Due to the increased reliance of disabled people on their mobile devices, ensuring that app features are accessible has become more important than ever before. Awareness as to the accessibility of apps has been growing as well. Google and Apple (primary organizations facilitating the app marketplace) have released developer and design guidelines for accessibility [7, 12]. They also provide accessibility services and scanners as part of their platforms [6, 11, 13, 42]. Mandates from government regulations, such as Section 508 and 504 of the Rehabilitation Act and the Americans with Disabilities Act (ADA) [2], are bringing further attention to accessibility factors in apps. Not surprisingly, as a result of such legislation, accessibility-related lawsuits in US federal courts have been growing, e.g., by 180% in 2018 compared to 2017 [46].

Despite these accessibility-focused efforts, studies have found significant accessibility issues in apps [40]. This suggests a continuing need for increasing accessibility awareness among researchers, developers of mobile platforms (e.g., Apple, Google), and developers of individual apps. Although some researchers have studied the accessibility of mobile apps, those studies remain limited in terms of the number of subjects considered, or the number of accessibility issues examined [3, 17, 18, 29, 40, 52]. Furthermore, it is not clear to what extent developers utilize accessibility features in their apps. To the best of our knowledge, no prior work has investigated the development practices pertaining to accessibility of mobile apps, to answer questions such as: how prevalent are different categories of accessibility issues in mobile apps? why developers write apps with accessibility issues? what do the developers want from the accessibility analysis tools? etc.

In this paper, we aim to cover this gap in research by providing a holistic view of Android accessibility from three complementary perspectives: apps, developers, and users. We investigate prevalence of accessibility issues (the apps), reasons why developers create apps with accessibility issues (the developers), and how accessibility issues impact user perception (the users). First, we conduct a mining study based on Android apps collected from AndroZoo [10] to investigate the extent to which accessibility issues are present. Next, we analyze the developers and organizations involved in the creation of these apps to determine their association with accessibility issues. We then conduct a survey with practitioners to gather a deeper understanding of the underlying reasons for creating apps with accessibility issues. Finally, we analyze user-provided reviews of the collected apps to understand potential associations between accessibility issues and users' perceptions.

Overall, the paper makes the following contributions:

- We report on the first large-scale analysis of prevalence of a wide variety of accessibility issues (11 types) in over 1,000 Android apps across 33 different application categories.

- We present the findings of a survey involving 66 practitioners, which shed light on the current practices and challenges pertaining to accessibility, as well as practitioners' perception regarding accessibility tools and guidelines.
- We discuss how the presence of accessibility issues, and their extent, impact users' perception of apps.
- Based on our results, we outline implications for developers and researchers, and provide suggestions for improving the existing tools to better support accessibility in mobile apps.

The paper is structured as follows: we provide a background on accessibility issues in Android in Section 2, followed by a brief review of prior research efforts in Section 3. In Section 4, we present our approach of mining Android apps and surveying developers for answering our intended research questions. In Section 5, we present our findings. Section 6 discusses the results and outlines implications for developers and researchers.

2 ACCESSIBILITY ISSUES IN ANDROID

Accessibility is defined as *"The quality of being easily reached, entered, or used by people that have a disability"* [27]. Mobile accessibility refers to making websites and apps more accessible to people with disabilities when using smartphones and other mobile devices [48]. Various mobile accessibility standards have been proposed, including W3C [48], Web Content Accessibility Guidelines (WCAG 2.0 and 2.1) [50], U.S. Revised Section 508 standard [1], and BBC Standards and Guidelines for Mobile Accessibility from the UK [15]. Within these standards, variety of recommendations have been made to provide better support for individuals with different kinds of disability including motor, hearing, and visual impairment. Several companies have also created their list of developer guidelines based on standards such as Android Accessibility Developer Guidelines [7], Apple Accessibility Developer Guidelines [12], and IBM Accessibility Checklist [26]. In this study, we focus on accessibility issues in Android apps and consider the accessibility recommendations provided by Google for Android developers [7].

Here we present a brief description of accessibility issues within Android apps that are the focus of our study.

2.1 Content Labeling

Impacted audience: Individuals with visual impairment.
Description: Content labels are alternative texts to images/actions. Although they are invisible, they can be accessed and announced by a screen reader (e.g., *TalkBack* in Android). They are intended to provide a clear description of images or actions of buttons for individuals with visual impairment. Below is a list of issues related to content labeling.

- Speakable Text: This issue indicates User Interface (UI) elements that are visible on the screen, but missing text labels. Presence of this accessibility issue means screen reader will be unable to convey these elements to visually impaired users. This issue can be fixed by providing certain attributes in the layout XML file, or dynamically in the code.
- Duplicate Speakable Text: This issue indicates UI elements with the same labels are visible in the same screen. Duplicate labels make it difficult for the user to separate and identify each UI element.
- Redundant Description: For native Android elements such as a `Button`, screen readers can access the type of UI element and announce it to the user along with the label. Repeating the type of element in the label is redundant and may confuse the user.

2.2 UI Implementation

Impacted audience: Individuals with mobile impairment.
Description: Developers are required to avoid certain implementations of UI elements that challenge users with mobile impairment. Below is a list of accessibility issues related to UI implementation.

- Clickable Span: UI elements such as `ClickableSpan` may not be compatible with accessibility services such as screen readers, i.e., hyperlinks within those elements may not be detectable and will not be activated. For these hyperlinks to be supported by accessibility services, the use of alternative UI elements such as `URLSpan` is encouraged
- Duplicate Clickable Bounds: This issue indicates two or more elements, such as `nested Views`, that share the same space and boundaries on the screen. When using alternative navigation approaches, duplicate views can cause the same area on the screen to be focused more than once.
- Editable Content Description: This issue highlights improper setting of properties when implementing `EditText` elements. Supporting accessibility for `EditText` requires implementing a label describing the field when it is empty. Moreover, once the user enters text, that text should be announced.
- Unsupported Class Name: Native Android `View` provides type information to screen readers and other accessibility services. For example, accessibility services can recognize an element as Button or Checkbox automatically and announce that to the user. However, developers sometimes create a custom `View`, but forget to provide this information to screen readers and other accessibility services. This issue highlights a custom `View` that does not provide the type of elements to screen reader.
- Traversal Order: Screen readers navigate the elements on a screen based on their hierarchical order. Developers can override this navigation order by using specific attributes for each UI element within the XML layout file, allowing them to specify the following and previous elements on the screen. This issue identifies cases where cyclic navigation is present, which may leave the user stuck at a certain element and unable to explore the remaining elements on the screen.

2.3 Touch Target Size

Impacted audience: Individuals with mobile impairment.
Description: Small targets are difficult to tap accurately. This requires more effort for the user. Failure to successfully tap on a button may impede using the app altogether.

- Touch target size: This issue identifies clickable UI elements with small touch areas that can be difficult to use.

2.4 Low Contrast

Impacted audience: Individuals with visual impairment.
Description: Insufficient contrast among an app's UI elements can affect how easily users can read, find, and comprehend those elements. Below is a list of accessibility issues related to contrast.

- Text Contrast: This issue corresponds to visible text, where there is a low contrast ratio between the text color and background color.

- Image Contrast: This issue identifies images that are visible on the screen but with a low contrast ratio between the foreground and background colors.

3 RELATED WORK

Although there has not been a prior large-scale analysis of accessibility in mobile apps, related work has been performed on the conventional desktop and web applications. Several studies for testing compliance of various websites have found that lack of accessibility is still a significant issue [3] [18] [29] [52]. Our current work builds upon and is informed by prior research on mobile accessibility implementation.

3.1 Previous Empirical Studies

Recently, studies have started to focus and further investigate accessibility issues in mobile apps. However, the number of these studies is in comparison smaller than those for the web. Notably, most of the prior work in this space is small scale. Coelho et al. manually evaluated four government mobile apps using W3C Accessibility Guidelines [50] and found that accessibility issues are extensive in these cases [43]. Milne et al. investigated the accessibility of mobile health sensors for blind users [33]. Walker et al. evaluated weather apps and found them not to be universally accessible [49].

Vendome et al. [17] performed an empirical study to understand Android apps' accessibility issues. However, our study is answering a much wider and more comprehensive set of questions, as we take a broader approach by looking into the perspective of users, developers, and apps, while their study only investigates the developer's perspective. Furthermore, our study analyzes 10 additional types of accessibility issues. Finally, our approach employs dynamic analysis for detection of accessibility violations, while they use static analysis, enabling us to detect a wider variety of accessibility issues as several UI elements in Android are populated at runtime.

Researchers have also previously looked at specific accessibility issues, such as alternative text labels [41], alternative image labels [33, 35, 43], and missing labels [35]. These studies help characterize accessibility problems. However, the small scales at which they were performed make it difficult to more generally assess the state of accessibility in mobile apps. Additionally, none of these studies has considered developers and attempted to understand the development practices that lead to the occurrence of accessibility issues in apps. Our study aims to fill these gaps.

3.2 Tools for Assessing Accessibility

Accessibility evaluation can be a complex task, requiring human expertise and judgment to provide an accurate assessment. However, certain facets of accessibility testing can be automated. Despite several studies on mobile accessibility evaluation, only a few evaluation tools are available. These tools belong to two primary categories: static analysis and dynamic analysis. Lint [31] is a static analysis tool that runs as part of the SDK but also integrated with the Android Studio IDE. It reports missing content descriptions and missing accessibility labels declared directly in the XML layout files. Dynamic analysis has the ability to identify and detect more accessibility issues [20]. Accessibility Scanner [42] is one of the dynamic analysis tools available on Google Play Store. The tool is based on the Accessibility Testing Framework, an open-source library of various automated checks for accessibility. Espresso [21] and Robolectric [39] are also general-purpose testing frameworks that can evaluate the same sets of accessibility issues as the Accessibility Scanner, since they are all based on the same library. PUMA [25] is a customizable dynamic analysis framework that can be used to check apps for different properties such as performance, security, and correctness. The tool also provides accessibility checks for different types of accessibility issues.

Patil et al. proposed an improved version of UI Automator Viewer, named "Enhanced UI Automator Viewer" [37], which is a tool for scanning and analyzing the UI components of an Android application. This tool supports inspection of low color contrast accessibility issue. MATE is another automated accessibility testing framework for Android that explores different app screens at runtime [20]. Mobile Accessibility Checker (MAC) is a tool from IBM for accessibility evaluation [54]. Krainz et al. proposed a model-driven development framework with automated code generation to avoid accessibility issues [30]. Ross et al. proposed an epidemiology-inspired framework to support the assessment of mobile apps accessibility [40].

All in all, none of the aforementioned approaches deals with the main objectives of our work: gaining a holistic view regarding the prevalence of accessibility issues, the reasons why developers create apps with accessibility issues, and how these accessibility issues impact user perception.

4 METHODOLOGY

Our study consisted of the following steps: (1) we first collected a large set of Android apps and filtered those that were not buildable; (2) we evaluated the accessibility of subject apps using a custom-build tool that we developed on top of popular accessibility libraries and testing frameworks; (3) we collected and analyzed developer and organization information pertaining to each app to identify their association with accessibility issues; (4) we then conducted a survey with practitioners to gather a deeper understanding of the underlying reasons for developing apps with accessibility issues; and finally (5) we manually analyzed user-provided reviews of the collected apps to understand potential associations between accessibility issues and users' perception. We now describe each of these steps in further detail.

4.1 Study Subjects

For our study, we selected Android as it is the most popular mobile platform [9]. We selected 1,500 top free apps from Google Play Store. These apps belonged to 33 different categories such as health and fitness, music and audio, productivity, and etc. After identifying the apps, we downloaded their APKs from AndroZoo [5], which is a repository of Android apps with more than 9 million APKs. As part of our study, we wanted to investigate if same developers tend to create similar types of accessibility issues. We selected a random set of 60 developers and found that 52 of them had multiple apps among the initial 1,500 apps. We then identified other apps from Google Play Store that these developers have published and are not already in our list of projects. This criteria added 200 more apps to our list.

Since one of our goals was to investigate how accessibility levels change over time, we needed to analyze multiple versions of apps. We selected the top 60 apps in terms of their *Activity coverage* (defined in Section 4.3) and obtained multiple versions for each app.

4.2 Accessibility Evaluation Tool

To evaluate the accessibility features of Android apps, we developed an accessibility evaluation tool that leverages the accessibility checks provided by Google's Accessibility Testing Framework [23], which is an open-source library that supports various accessibility-related checks and can be applied to various UI elements such as `TextView`, `ImageView`, and `Button`. Google's Accessibility Testing Framework is also the underlying engine that is used by Google Scanner [42], a Google recommended app for assessing the accessibility of Android apps. We did not use Google Scanner [42] for our study, as it requires the users to manually run the app, go through each screen and initiate the evaluation process, making it time-consuming and not scalable for a large-scale analysis. Since Google's Accessibility Testing Framework is open-source, it provided us with the opportunity to integrate it into our evaluation tool with ease and automate the entire process.

Our accessibility evaluation tool has two major parts. One part simulates user interactions and the other part monitors the device for `Accessibility Events`. We detail each part below.

4.2.1 Simulating User Interactions. We assess accessibility of apps dynamically, as several UI elements in Android are populated at runtime, making it rather difficult to detect them statically. To that end, our tool first installs the app on an emulator running on a laptop with Intel Core i7-8550U, 1.80GHz CPU, and 16GB of RAM. We used an Android image configured with Google services, API level 25 and 1080 by 1920 pixel display resolution.

After successfully installing an app on the emulator, our tool uses Android Monkey [8] to simulate user interaction. Android Monkey [8] is a UI testing tool developed by Google. It generates pseudo-random gestures, such as clicks and touches, to simulate user interactions.

Our accessibility evaluation tool runs each app for a time limit of 30 minutes, during which the app is restarted multiple times to maximize the coverage of Activities and prevent Monkey from getting stuck on specific screens. In the case of a crash, the tool restarts the app and continues to crawl. Monkey takes a value as the seed to generate the random events. We feed Monkey with a different seed value for each run to maximize coverage. Additionally, at this step, we collect coverage metrics, such as the number of covered Activities and lines of code.

4.2.2 Monitoring Accessibility Events. We developed an Android app, called *Listener*, that was installed on the emulator as part of our accessibility evaluation tool. Listener has a Service running in the background that uses Android's `Accessibility API` to listen for `Accessibility Events`, as each app is crawled. `Accessibility API` is included in Android to support the implementation of accessibility services. `Accessibility Events` are system-level signals that indicate state changes on the device, e.g., when a Button is clicked, or a new screen is opened. Every time an `Accessibility Event` is detected, the app takes a screenshot of the current screen, and retrieve the hierarchy of all the UI elements (Views) that are visible to the user. It then invokes Google's Accessibility Testing Framework to perform the various accessibility checks [23]. Since there are no benchmarks for evaluating this kind of tool, we evaluated the tool by running it on several apps and manually verifying the results.

4.3 Data Collection and Analysis

We collected different types of accessibility issues for each app using our accessibility evaluation tool. We also collected Package name, Activity name, and number of user interface elements and lines-of-code for each app. We calculated the app `Activity coverage` by dividing the number of unique Activities that are explored by the total number of Activities in the app. We eliminated apps from our analysis with very low `Activity coverage`, i.e., apps for which our tool was not able to explore more than one Activity. We finally ended up with 1, 135 apps in our corpus. Figure 1 shows the distribution of apps in our final dataset.

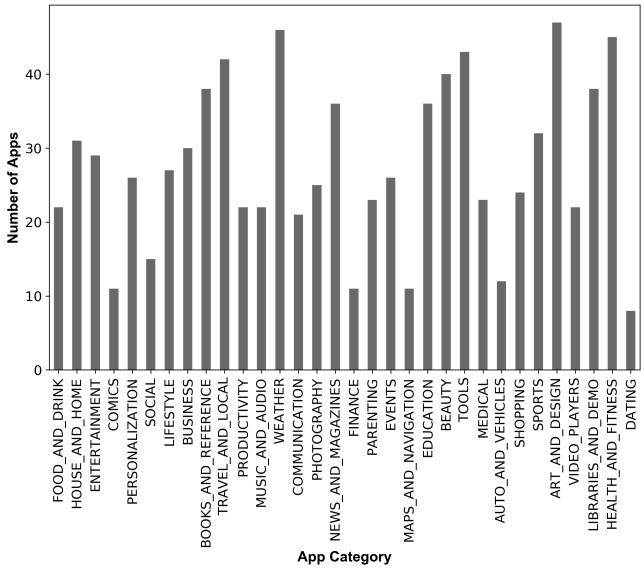

Figure 1: Number of apps for each category in the dataset

For apps in our dataset that were obtained from Google Play Store, we crawled each app page and collected various meta-data including category, name of developer, number of installs, number of reviews and rating score. Since we collected accessibility issues from screens, and screens with larger number of elements are prone to more accessibility issues, we needed to normalize the data to avoid such bias. To that end, we used *inaccessibility rate*, a metric calculated by dividing the number of elements with accessibility issues on a screen over the total number of elements on the same screen that are prone to accessibility issues [54]. This ratio is calculated for each of the 11 types of accessibility issues. For example, the inaccessibility rate for *TextContrast* type would be the number of elements with *TextContrast* issues divided by all the *TextView* elements that are potential victims for this type of accessibility issue.

Game UIs are not built using native UI elements, instead they are built based on graphic libraries such as OpenGL [24] and Unity [47] [14], where interactive UI elements such as buttons are rendered as images in the background. Existing tools can neither evaluate these elements nor examine their properties, since these elements do not provide enough information to the accessibility framework. As a result, we excluded the Game category from our analysis.

An important concern with existing accessibility analysis tools is that they report all accessibility issues without assigning any

ranking. Our goal was to identify fruitful ways in which the accessibility issues could be ranked, thus engineers can prioritize their effort in resolving such issues. To identify plausible approaches, we randomly selected 25 apps and 100 screens from our dataset, and manually analyzed the reported accessibility issues. Our manual analysis revealed three different cases: (1) Some accessibility issues make it difficult to use the app, while others make the app completely unusable. As an example the 5-star rating element of the Yelp app lacks accessibility support, leaving this core functionality inaccessible to many people. This led us to define *severity of impact on the user* as a ranking criterion. (2) Some accessibility issues are easily fixed, while others require redesigning the interface completely. This motivated us to define *ease of fix* as another ranking criterion. (3) Not all reported accessibility issues are in fact accessibility issues, as the tools sometimes produce false positives. Thus, we defined *certainty of the warning (true positive)* as another plausible ranking criterion.

We also investigated the impact of company culture on accessibility issues. We identified apps in our dataset that are developed by well-known companies. It is naturally the expectation that such companies would have better software development resources than others. The selection criteria is based on the Forbes Top 100 Digital Companies list [22]. The list contains companies such as Amazon, Google, and Microsoft. 23 apps met this criteria. We also analyzed the impact of accessibility issues on users perception. To answer this question, we crawled the Google Play Store and collected metadata about each app including app rating score, and whether it was promoted as an Editors' Choice on the Store. Our analysis covered reviews written in English only. Prior to performing the review analysis, we pre-processed the text of the reviews using NLTK library [34]. We applied text tokenization, stemming, and lower-case letters conversion. We then searched the dataset using a set of accessibility-related keywords that are based on the different accessibility guidelines and tools, sample keywords include "accessibility", "visual impairment", "blind", etc. We improved upon this set as we scanned through reviews that discussed accessibility. The search process flagged 704 reviews. Two authors independently read the reviews and assigned the accessibility concern types and whether the sentiment was positive or negative. We also had a high inter-rater reliability of 0.84. . We ended up with 150 verified accessibility-related reviews from 102 different apps.

4.4 Survey

To validate our findings, we performed an online survey of Android developers. In this section, we describe the survey design, participant selection criteria, pilot survey, data collection, and analysis.

4.4.1 Survey design. We designed an online survey to gather a deeper understanding of the underlying reasons for creating apps with accessibility issues. We asked demographic questions to understand the respondents' background (e.g., their number of years of professional experience). We then asked them about their current practice of using guidelines and tools for assessing accessibility (if any). We also asked them about the challenges of ensuring accessibility based on their experiences. We presented some of the accessibility challenges identified through our empirical analysis of apps and asked the respondents to rate each of them with one of the following ratings and to provide a rationale for their rating: *very important, important, neutral, unimportant, very unimportant.* A respondent can also specify that he/she prefers not to answer. We included this option to reduce the possibility of respondents providing arbitrary answers. During our app analysis, we noticed that none of the available accessibility analysis tools distinguish between the reported accessibility issues. We identified three potentially fruitful methods of ranking reported accessibility issues: *severity of impact on the user, certainty of the warning (true positive), and ease of fix.* We asked the respondents to rate each of the 3 ranking methods with one of the following ratings and to provide a rationale for their rating: *very important, important, neutral, unimportant, very unimportant.* A sample of the survey instrument can be found at the companion website [44].

4.4.2 Participant Selection. We recruited participants for the survey from the list of open-source app developers on F-Droid. In total, we identified 740 unique email addresses for our survey.

4.4.3 Pilot Survey. To help ensure the validity of the survey, we asked Computer Science professors and graduate students (two professors and two Ph.D. students) with experience in Android development and in survey design to review the survey to ensure the questions were clear and complete. We conducted several iterations of the survey and rephrased some questions according to the feedback. In this stage, we also focused on the time limit to ensure that the participants can finish the survey in 10 minutes. The responses from the pilot survey were used solely to improve the questions and were not included in the final results.

4.4.4 Data Collection. We used Qualtrics [38] to send a total of 740 targeted e-mail invites for the survey. 6 of those emails bounced and we received 9 automatic replies, leaving at most 725 potential participants, assuming all other emails actually reached their intended recipients. According to the Software Engineering Institute's guidelines for designing an effective survey [28], *"When the population is a manageable size and can be enumerated, simple random sampling is the most straightforward approach"*. This is the case for our study with a population of 740 software developers.

From the 740 sent emails, we received 66 responses (8.9% response rate). Previous studies in software engineering field have reported response rates between 5.7% [36] and 7.9% [32]. We disqualified 5 partial responses. Finally we considered 61 responses. We received responses from 18 countries across 5 continents. The top two countries where the respondents reside are Brazil and the United States. The professional experience of our respondents varies from 0.25 years to 6 years, with an average of 3.11 years.

4.4.5 Data Analysis. We collected the ratings our respondents provided for each accessibility issue, converted these ratings to Likert scores from 1 (Strongly Disagree) to 5 (Strongly Agree) and computed the average Likert score. We also extracted comments and texts from the "other" fields by the survey respondents explaining the reasons behind their choices. To further analyze the results, we applied Scott-Knott Effect Size Difference (ESD) test [45] to group the accessibility issues into statistically distinct ranks according to their Likert scores. We excluded responses that selected "I don't know" for our ESD test. Tantithamthavorn et al. [45] proposed ESD as it does not require the data to be normally distributed. ESD leverages hierarchical clustering to partition the set of treatment

means (in our case: means of Likert scores) into statistically distinct groups with non-negligible effect sizes.

5 RESULTS

In this section, we present the results of our study from three complementary perspectives: apps, developers, and users.

5.1 App Perspective

We start by looking into the prevalence of accessibility issues in the apps. More specifically, we answer the following research question.

5.1.1 ***RQ1: How prevalent are accessibility issues in Android apps?*** We measure the *inaccessibility rate* of each app for the 11 types of accessibility issues explained earlier in Section 2. The inaccessibility rate is calculated by dividing the number of UI elements (such as *TextView* or *Button*) infected with accessibility issues by the total number of UI elements that are prone to such accessibility issues. We also use the *overall inaccessibility rate*, which is the average of all the inaccessibility rates for the different types of accessibility issues.

Figure 2 shows the distribution of overall inaccessibility rate for all the apps in our dataset. As shown in figure 2, a small number of apps have no accessibility issues, while most apps do. In our dataset, the mean inaccessibility rate for each app is 6.04% and the standard deviation is 2.42%.

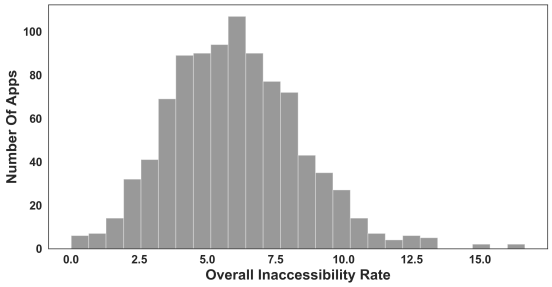

Figure 2: Distribution of inaccessibility rate among apps

Observation 1: Accessibility issues are prevalent across all categories of apps, and the mean *inaccessibility rate* is 6.04%.

5.1.2 ***RQ2: What are the most common types of accessibility issues? Are specific categories of apps more susceptible to accessibility issues than others?*** Next, we take a closer look at the inaccessibility rate among the various types of accessibility issues. Table 1 shows that *Text Contrast*, *Touch Target*, *Image Contrast*, and *Speakable Text* are the most frequent and have a mean of 22.81%, 19.78%, 12.85%, and 11.08%, respectively. Almost a quarter of *TextView* elements reported a *Text Contrast* issue. None of the apps in our dataset had a *Traversal Order* accessibility issue. Since this is a very specific problem with app navigation approach that is optional to use by developers, zero occurrence of this accessibility issue is not surprising. We omit this issue from our analysis in the rest of the tables.

Figure 3 shows inaccessibility rate of the apps grouped into 33 categories. We observe that despite slight variations, all categories have accessibility issues. The overall inaccessibility rate is between 4.2% and 7.3.%. Music and Audio category exhibits the highest inaccessibility rate of 7.3%. While different categories of apps have

Table 1: The distribution of accessibility issues

Type of accessibility issue	Mean	Std	Max
TextContrast	22.81	11.61	65.10
TouchTargetSize	19.78	10.09	52.63
ImageContrast	12.85	11.99	50.0
SpeakableText	11.08	8.34	42.24
RedundantDescription	0.93	3.40	50.0
DuplicateSpeakableText	0.89	1.47	15.45
ClassName	0.68	1.96	19.64
DuplicateClickableBounds	0.55	0.92	8.33
EditableContentDesc	0.31	2.69	50.0
ClickableSpan	0.15	0.95	14.72
TraversalOrder	0.0	0.0	0.0
All accessibility types	6.04	2.42	16.64

similar distribution of overall inaccessibility rates, certain types of accessibility issues are more frequent in some categories. For example, apps in the *Finance* category have the lowest *SpeakableText* inaccessibility rate at about 4.0%, while *Design and Beauty* has the highest inaccessibility rate of this type at around 16.0%.

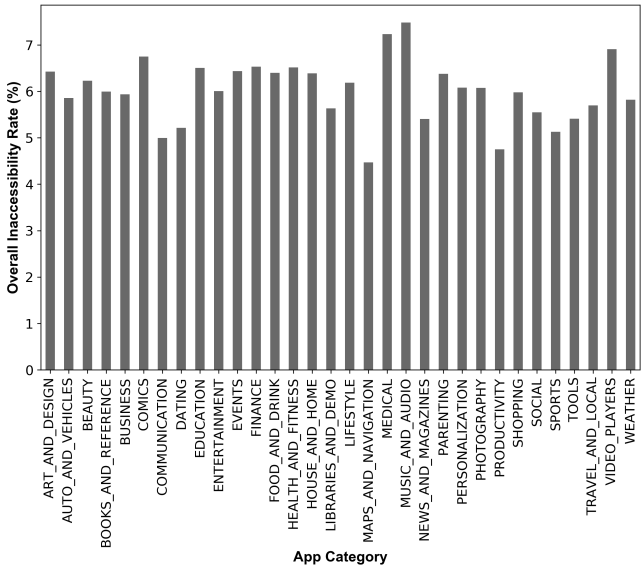

Figure 3: Distribution of inaccessibility rates across the different categories

Observation 2: 10 out of 11 types of accessibility issues evaluated in the study were present in the evaluated apps, but to varying degrees.

Since developers use templates provided by Android Studio to build their apps, we posit that the presence of accessibility issues in templates can contribute towards the prevalence of accessibility issues in apps. To that end, we analyzed the templates and found that 5 out of the 10 templates provided by Android Studio suffered from *Text Contrast* , *Touch Target* and *SpeakableText* accessibility issues. For instance, a screen built using *Tabbed Activity* template has *Text Contrast* issues for the titles of the different tabs (as shown in Figure 4).

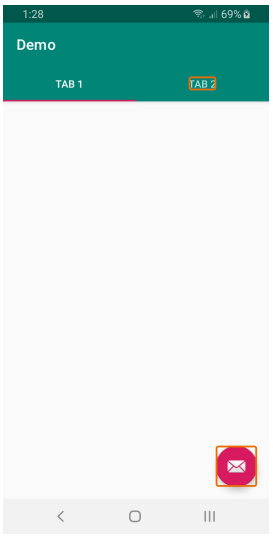

Figure 4: Two accessibility issues are identified in the *TabbedActivity* template. *TextContrast* for the title of inactive tab, and a missing *SpeakableText* For the *Button*

Observation 3: 50% of the templates provided by Android Studio, the most popular IDE for Android development, have accessibility issues.

5.1.3 ***RQ3: How does accessibility evolve over time in Android apps?*** In order to answer this question, we used a subset of the apps with multiple versions (details in Section 4). We excluded apps with only one version from our analysis as our goal was to investigate the evolution of accessibility issues. In total, this analysis involved 60 apps with 181 versions. We then performed the accessibility evaluation and calculated the difference in inaccessibility rate among the subsequent versions. A positive difference indicates more accessibility issues than the prior version, and vice versa.

Since we are using inaccessibility rate instead of the total number of accessibility issues for our analysis, changes in the app user interface are less likely to impact the results. Figure 5 depicts the summary of changes in accessibility issue for 128 updates for 53 apps. Majority of the updates (47%) improved the app's overall accessibility, 28% of the updates impacted the overall accessibility negatively, and for the remaining 25% overall accessibility levels remained the same. Note that despite the use of inaccessibility rate, it is possible that the reduction in inaccessibility rate is not due to fixes but due to the addition of UI elements. However, we still consider it an improvement in the overall accessibility of a new version, as the new UI elements did not introduce any new accessibility issues.

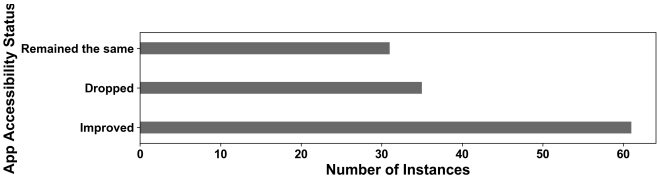

Figure 5: How Apps accessibility levels changed over time

Observation 4: Apps become more accessible over time, with 47% of app updates improving the overall accessibility.

5.2 Developer Perspective

We now explain our findings regarding the associations between developer/organization and accessibility issues.

5.2.1 ***RQ4: Do same developers tend to create similar types of accessibility issues?*** We examine whether developers are creating apps with similar types of accessibility issues. To answer this question, we first identified a subset of developers who had contributed to multiple projects in our corpus. We then explored Google Play Store to identify all apps written by these developers, which yielded 200 new apps. Finally. we calculated the inaccessibility rate for 260 apps. Table 2 shows the average Standard Deviation (SD) of inaccessibility rate for apps developed by the same developer and apps developed by different developers. In the first column of Table 2, first we calculate the standard deviation for apps grouped by each developer who has multiple apps and then calculate the average. In the second column, we did a similar calculation but only for developers who contributed a single app. From table 2, we observe that the average standard deviation of inaccessibility rate in apps developed by the same developers is 1.70, whereas it is 2.42 for apps developed by different developers.

We performed a Two Sample t-test for each category of accessibility issues and found that for three of the categories (*SpeakableText*, *DuplicateSpeakableText* and *ClassName*) population means are statistically significant (Two Sample t-test, $p < 2.2 \times 10^{-16}$) represented using an asterisk (*) in Table 2. Since we are performing multiple tests, we have to adjust the significance value accordingly to account for multiple hypothesis correction. We use the Bonferroni correction [19], which gives us an adjusted α value of 0.004 to be used as the significant level. We also report the Cohen's d value (effect size) for each accessibility issue. A Cohen's d value ≥ 0.8 indicates a large effects size, a value ≥ 0.5 and < 0.8 indicates a medium effects size, and a value ≥ 0.2 and < 0.5 indicates a small effects size. Although the observed effect is for the most part small, it is not negligible. This is reasonable, because other factors also impact the presence of accessibility issues.

Table 2: Comparison of SD values for apps made by the same developers, and different developers.

Accessibility issue	Inaccessibility rate SD		Cohen's d
	Same developer	Different developers	
TextContrast	9.35	11.61	0.02
TouchTargetSize	7.86	10.09	-0.14
ImageContrast	7.64	11.99	0.06
SpeakableText *	4.50	8.34	0.33
RedundantDescription	0.66	3.40	0.02
DuplicateSpeakableText *	0.69	1.47	-0.23
ClassName *	0.42	1.96	-0.31
DuplicateClickableBounds	0.40	0.92	0.14
EditableContentDesc	0.0	2.69	-0.18
ClickableSpan	0.76	0.95	0.09
Overall inaccessibility	1.70	2.42	0.01

Observation 5: App developers tend to create apps with similar types of accessibility issues.

5.2.2 ***RQ5: What are the underlying reasons for developing apps with accessibility issues?*** We surveyed Android developers (See Section 4 for details) to get their opinion about the reasons behind accessibility issues in apps.

Table 3 represents the percentage of respondents selecting an option. It shows that *lack of awareness about accessibility and its importance* is identified as the top reason (48.53%). *Additional cost* and *lack of support from management* are the other top reasons.

Table 3: Reported challenges with ensuring accessibility

Challenges with Ensuring accessibility	Percent
Lack of awareness about accessibility and its importance	48.53
Additional cost of ensuring accessibility	16.50
Lack of support from management	15.53
Lack of tools	9.70
Lack of standards and guidelines	8.73
Not sure which standards to follow	0.97

Observation 6: Developers perceive lack of awareness as the top reason for introducing accessibility issues in apps.

5.2.3 ***RQ6: Do apps developed by large and well-known companies have better inaccessibility rate than other apps?*** To answer this question, we identified apps that are developed by well-known companies such as Amazon, Google, Microsoft, and etc. The selection criteria is based on the Forbes Top 100 Digital Companies list [22]. We posit that these companies have access to more experienced developers and better development resources than others. 23 apps satisfied this criterion. From table 4, we observe that the mean for one type of accessibility issue is noticeably different. *SpeakableText* rate is 70% lower in the apps produced by top companies compared to the mean for all other apps in the dataset. Surprisingly, no major difference is observed among the other accessibility issues.

Table 4: Comparison of the mean for apps' inaccessibility rates developed by top companies against all other apps

Accessibility issue	Inaccessibility rate		Cohen's d
	Apps by top companies	Other apps	
TextContrast	20.63	22.75	-0.28
TouchTargetSize *	19.79	19.67	0.01
ImageContrast	12.60	12.86	0.02
SpeakableText *	3.39	11.21	-1.14
RedundantDescription *	2.49	0.90	0.36
DuplicateSpeakableText	1.10	0.89	0.12
ClassName *	1.68	0.68	0.47
DuplicateClickableBounds	0.33	0.55	-0.28
EditableContentDesc *	1.80	0.31	0.37
ClickableSpan *	0.76	0.14	0.41
Overall inaccessibility *	4.80	6.03	-0.48

We wanted to understand the reason behind this. In the survey, we asked the developers whether accessibility evaluation is part of their app development/testing process. Out of the 32 survey respondents that are paid developers, 23 did not have any accessibility evaluation as part of their app development process. We also asked the developers whether accessibility of their apps is treated with importance in their organization. 27 respondents mentioned that accessibility is not treated as importantly as other quality attributes, such as security, in their organization. These might be some of the reasons why apps developed by top companies are as susceptible to accessibility issues as apps developed by other companies.

Observation 7: The inaccessibility rates for apps developed by top companies are similar to inaccessibility rates for other apps, except for *SpeakableText* accessibility issue.

5.2.4 ***RQ7: Do developers perceive all accessibility issues equal?*** Our goal was to understand how developers perceive different accessibility issues. To that end, we presented a list of some of the accessibility issues identified through the app analysis and asked the developers to rate them (See Section 4 for details). Table 5 presents the 6 accessibility issues ranked according to the Scott-Knott ESD test in terms of means of Likert scores for all the respondents. *Redundant Description* , *Text Contrast* and *Image Contrast* are the top two groups.

Table 5: Accessibility issues ranked according to the Scott-Knott ESD test (all respondents)

Group	Accessibility issue category
1	RedundantDescription
2	TextContrast
2	ImageContrast
3	TouchTargetSize
3	DuplicateSpeakableText
4	SpeakableText

In our study, we noticed that none of the existing tools rank the reported accessibility issues, thus do not provide any means of prioritizing accessibility issues that should be resolved by the developer. We asked the respondents to rate three ranking methods that we identified through a manual analysis and provide a rationale for their rating (See Section 4 for details). We used Scott-Knott ESD test to rank their responses. Respondents ranked *severity of impact on the user* as the primary criterion, and *certainty of the warning (true positive)* and *ease of fix* as the second and third criterion, respectively.

Observation 8: Developers believe *impact on user* should be the primary criterion for ranking (prioritizing) the accessibility issues.

5.3 User Perspective

Here, we report our findings regarding how users' perception about apps is affected by the presence of accessibility issues.

5.3.1 ***RQ8: What accessibility issues do users complain about?*** To answer this question, we identified 704 reviews from 102 different apps using the process explained in Section 4, resulting in 150 accessibility-related reviews. Figure 6 summarizes the content of all the app reviews in terms of the type of accessibility concern discussed. Almost half of the reviews discussed accessibility without specifying the exact issues. Users in the other half of the reviews discussed mainly 3 concerns: (1) difficulties related to missing label or content description, (2) text size or color, and (3) image/icon contrast or size. We also found that users tend to use app review to communicate both positive and negative experiences with app accessibility. In some cases users gave a bad review and stated that they are going to delete the app as it is unusable.

Some of the accessibility issues that were discussed in the reviews are not detectable by automated tools and require manual evaluation. For example, users reported issues related to the grouping of

UI elements on the screen. Users with visual impairment may rely on using Google TalkBack linear navigation service to understand what is shown on the screen (they swipe right and left to move from one element to another). In one app screen, there were too many elements visible, and navigating that screen linearly was a tedious task, and slowed down the reading experience. Alternatively, developers should hide elements that do not add value to the user, either by marking them as unimportant for the screen reader, or by grouping them with other UI elements under descriptive headings.

Figure 6: The rate of the different accessibility concerns discussed in app reviews

Observation 9: Almost half of the accessibility issues reported by users in app reviews are about difficulties related to missing label or content description, text size or color, and image/icon contrast or size.

5.3.2 ***RQ9: Do accessibility issues have any association with app ratings?*** We explore the association, if any, between apps' inaccessibility rate and their user ratings on Google Play Store. We crawled the Google Play Store and collected meta-data about each app including user rating, and whether it was promoted as an Editors' Choice on the Store. Table 6 shows the details of the correlation analysis for apps' inaccessibility rates and user ratings. We used Pearson correlation coefficient since the data is normally distributed.

Table 6: Inaccessibility rates correlation with app rating

Accessibility issue	Correlation value
TextContrast *	0.150
TouchTargetSize	-0.023
ImageContrast *	0.108
SpeakableText	-0.020
RedundantDescription	0.019
DuplicateSpeakableText	-0.059
ClassName	-0.028
DuplicateClickableBounds	0.012
EditableContentDesc	-0.020
ClickableSpan	0.022
Overall inaccessibility rate	0.050

Observation 10: There is no strong association between the presence of accessibility issues and app ratings.

We also checked whether presence of inaccessibility issues has association with being promoted as Editors' Choice. Table 7 compares the inaccessibility rate for apps that were promoted as Editors' Choice (a total of 83 apps in our dataset) against all other apps that were not promoted as Editors' Choice. Interestingly, apps that were selected as Editors' Choice had similar inaccessibility rate compared to those that were not selected.

Observation 11: Presence of accessibility issues does not impact popularity of an app.

Table 7: Comparison of inaccessibility rates for apps that were selected as Editors' Choice in Play store.

Accessibility issue	Inaccessibility rate		Cohen's d
	Editors Choice	Other apps	
TextContrast	25.30	22.71	0.23
TouchTargetSize	18.99	20.19	-0.12
ImageContrast *	15.41	12.43	0.25
SpeakableText	9.81	11.16	-0.16
RedundantDescription	1.12	0.97	0.04
DuplicateSpeakableText	0.71	0.91	-0.15
ClassName	1.04	0.66	0.18
DuplicateClickableBounds *	0.37	0.59	-0.26
EditableContentDesc	0.18	0.36	-0.07
ClickableSpan	0.05	0.17	-0.16
Overall inaccessibility	6.29	6.02	0.11

6 DISCUSSION

Accessibility issues are widely prevalent. One goal of our study was to investigate and identify the most prominent accessibility issues. Our findings show *Text Contrast*, *Image Contrast*, and *Touch Target* are widely prevalent accessibility issues in all categories (33 in our dataset) of Android apps. Since the apps used in this study are the top free apps in the Google Play Store, our results represent the accessibility status of the most commonly used Android apps on the market.

Individuals with different kinds of disability are affected. To make things worse, identified accessibility issues affect individuals with different types of disabilities. As an example, *Touch Target* accessibility issue impedes the app use for individuals with mobile impairment. Color- and image-contrast related accessibility issues create difficulty for visually impaired users. Our results also highlight the fact that approximately 39 million blind users and 246 million low-vision users worldwide [16] are mostly affected by accessibility issues, since the most frequent issues identified in our analysis were *Text Contrast*, *Image Contrast*, and *Touch Target*. These findings call for action to the software engineering community for reducing accessibility issues and lowering the barrier for individuals with different kinds of disability.

Accessibility issues are found even in the Android templates. Our study provided us with useful insights as to the underlying reasons for why existing app are so riddled with accessibility issues. One such reason appears to be the presence of accessibility issues in the templates provided by IDEs. 5 out of the 10 templates from Android Studio suffer from *Text Contrast*, *Touch Target* and *SpeakableText* accessibility issues (two of the top three identified accessibility issues). It is a common practice for developers to build their apps based on these templates. Interestingly, Android Studio is not only the most popular IDE for Android development, but it is also from Google, the company that makes the most popular accessibility analysis tool for Android (i.e., Google Accessibility Scanner).

Developer are generally unaware of the accessibility principles. The pervasiveness of accessibility issues can also be attributed to the lack of awareness and training among the developers. This was identified as the top challenge (48.53%) among the survey respondents (Table 3). For example, all of our survey respondents mentioned using Lint. However, prior research shows that developers do not actually know how accessibility warnings from Lint impact the

app's use by a disabled person [17]. Developers could benefit from training on accessibility design principles, yet such training is a rarity in most formal academic curricula.

Apps do not get worse over time. When we looked at the evolution of accessibility in apps, we found that 72% of the apps are either improving or remaining the same in terms of accessibility (Figure 5). This is surprising, for two reasons. First, conventional code quality attributes, e.g., code smells, tend to degrade over time as the complexity and size of apps increase [4], while with respect to accessibility we observe a reverse effect. Second, developers claim to be unaware of accessibility issues and standards (Table 3), thus the improvements in accessibility cannot be attributed to a conscious effort on their part. This warrants further investigation into why and how the accessibility of apps improve over time.

Tools have ways to go. Our results highlight the limitations of current accessibility analysis tools. Some of the accessibility issues are not easy to identify. One such example is the Dropbox app, which requires the user to navigate through all files in the current folder before accessing the"menu", "select", or "more" buttons. This makes it difficult, or even impossible, for individuals with mobile impairment to use the app. Identifying such accessibility issues is not straightforward and requires considering the context of interaction. None of the current accessibility analysis tools consider context. Existing tools are also unable to evaluate apps with non-native elements in their UIs such as Games. Attention from the research community is needed to investigate the accessibility issues that are context-specific or occur in non-native UI elements.

Furthermore, we found that the reported accessibility issues are not equally important. In some cases, a single inaccessible element undermines the entire purpose of an app. As an example, we found the rating UI widget of the Yelp app lacks accessibility support, leaving the core functionality of this app inaccessible to many users. We noticed that none of the analysis tools report severity of issues. Rather important issues are presented alongside of those that affect tangential functions of an app, e.g., text contrast on the About screen. The high number of accessibility issues reported by the existing tools, reaching hundreds in some cases, overwhelm the developers. Therefore, research is needed into identifying effective means of prioritizing accessibility issues in terms of their importance. We took the first step and identified three potentially fruitful methods of ranking accessibility issues: *severity of impact on the user, certainty of the warning (true positive), and ease of fix*. When asked to rank these during the survey, developers ranked "severity of impact on the user" as the primary criterion for prioritizing the accessibility issues. However, further research is required to identify and compare different prioritization criteria.

Another limitation of the existing accessibility testing tools is the inability to consider all kinds of impairment. As an example, Google Accessibility Scanner has no support for testing hearing impairment related issues. Devising new robust tools for automatically testing accessibility for all kinds of impairment requires further attention from the research community.

An interesting observation is that while accessibility issues can render an app completely useless for a disabled person, the fixes to many accessibility issues are in fact quite easy. For instance, text contrast can be easily fixed through simple changes to the font and/or background colors. Given the recent advances in automated program repair in the software engineering research community, this appears to be a fruitful avenue of future research.

Finally, one approach for stemming accessibility issues is to plan for accessibility in the early design phase rather than handling it as an afterthought at the end of the development phase. Catching accessibility issues early on at the design stage allows the developer to adapt the UI without significant effort. Automated accessibility analysis tools are needed that can be applied to early design prototypes, e.g., UI sketches. Furthermore, we found that the most popular build-time code scanning tool for Android, called *Lint*, only provides support for detection of one type of accessibility issue. We believe incorporating more comprehensive checks in the build process of IDEs could drastically reduce the accessibility issues that creep into the final product, allowing the developers to resolve the issues early in the development.

Gaps between developer and user perception. The preliminary indication from our study is that developers and users have different perceptions as to the impact of accessibility issues on the usability of apps. Bridging this gap would help developers prioritize fixes for accessibility issues that are most critical for users first. However, in order to do a meaningful comparison and have a better understating of user perspective, a more extensive user study, involving disabled users, would be needed. This is outside the scope of our current study, given that property conducting such a study would require, among others, access to users with different types of disability (e.g., visual, hearing, mobility impairment). Nevertheless, we consider this to be an interesting avenue of future work.

Organizations need to pay attention. When we tested the relationship between app ratings and presence of accessibility issues, we did not find any strong association. We posit this has to do with the fact that disabled people are a small minority of app users, thus, reports by this category of users do not have a significant impact on the overall app ratings. This became further evident when we saw a lack of association between presence of accessibility issues and popularity of an app (Table 7). Unfortunately, this implies that disabled users cannot rely on app ratings to determine which apps to install, and accessibility-related criticism of apps tends to go unnoticed, as most users do not share the same concerns.

Our analysis revealed that inaccessibility rates for apps developed by top companies are relatively similar to inaccessibility rates for other apps. Moreover, out of the 32 survey respondents that are paid developers, 23 did not have any accessibility evaluation as part of their app development process. Respondents also mentioned that accessibility is not treated as importantly as other aspects of quality, such as security, in their organization. Lack of support from management was also identified as one of the challenges (15.53%) with regards to ensuring accessibility (Table 3). All of these indicate that even the top companies in most cases do not pay attention to accessibility. We believe training the app developers and increasing their general awareness of the accessibility issues could improve the state of affairs, as they become ambassadors of accessibility in their organizations.

7 THREATS TO VALIDITY

We have strived to eliminate bias and the effects of random noise in our study. However, it is possible that our mitigation strategies may not have been effective.

Bias due to sampling : To increase our confidence that the subject apps are representative, we used multiple sources (i.e., Google Play Store and F-Droid). We also used large number of projects belonging to multiple categories. Since we used only two sources, our findings may not be generalizable to all Android apps. However, we believe that the large number of projects sampled from multiple sources adequately addresses this concern.

Bias due to tools used: Any error in the tools used may affect our findings. To minimize this risk, we leverage Google Accessibility Testing Framework which is widely used by other researchers [20, 40] and practitioners. This is also the underlying framework for the state-of-art Google Scanner.

Moreover, it is possible that there are defects in the implementation of our tool. To that end, we have extensively tested our implementation, among others, verifying the results against a small set of apps for which we manually verified the accessibility issues. Additionally, we do not claim to identify all accessibility issues, as it is possible that certain accessibility issues cannot be identified using the existing tools.

Bias due to survey: It is possible that the survey participants misunderstood some of the survey questions. To mitigate this threat, we conducted a pilot study with Android developers with different experience levels from both open-source community and industry. We also conducted a pilot study with survey design experts. We updated the survey based on the findings of these pilot studies.

8 CONCLUSION

We presented the results of a large-scale empirical study aimed at understanding the state of accessibility support in Android apps. We found accessibility issues to be rampant among more than 1,000 popular apps that were studied. We identified a number of culprits, including, among others, the observation that developers are generally unaware of the accessibility principles, and that existing analysis tools are not sufficiently sophisticated to be useful, e.g., are unable to prioritize accessibility issues. Due to the disproportionately small number of disabled users, apps with extensive accessibility issues are highly popular and have good ratings. Thus, disabled people have no way of determining which apps are suitable for their use based on the app ratings. Moreover, app developers appear to lack the incentives and backing of their organizations to make their apps usable by this small, yet important, segment of society.

Our ultimate goal is to help catalyze advances in mobile app accessibility by shedding light on the current state of affairs. Our findings can help practitioners by highlighting important skills to acquire, and educators by recommending important skills to include in the curriculum. The findings also highlight opportunities for researchers to address the limitations of existing tools.

The research artifacts for this study are available publicly at the companion website [44].

ACKNOWLEDGMENT

This work was supported in part by awards 1823262 and 1618132 from the National Science Foundation. The first author is supported by King Saud University for his PhD studies at the University of California, Irvine.

REFERENCES

[1] 508Standards. 2019. U.S. Revised Section 508 standards. https://www.access-board.gov/guidelines-and-standards/communications-and-it/about-the-ict-refresh/final-rule/text-of-the-standards-and-guidelines
[2] ADAlaws. 2019. ADAlaws. https://www.ada.gov/cguide.htm
[3] Gaurav Agrawal, Devendra Kumar, Mayank Singh, and Diksha Dani. 2019. Evaluating Accessibility and Usability of Airline Websites. In *Advances in Computing and Data Sciences*, Mayank Singh, P.K. Gupta, Vipin Tyagi, Jan Flusser, Tuncer Ören, and Rekha Kashyap (Eds.). Springer Singapore, Singapore, 392–402.
[4] Iftekhar Ahmed, Umme Ayda Mannan, Rahul Gopinath, and Carlos Jensen. 2015. An empirical study of design degradation: How software projects get worse over time. In *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on*. IEEE, 1–10.
[5] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*. IEEE, 468–471.
[6] AndroidAccessibility. 2019. Android Accessibility Overview. Accessed April 12th, 2018. https://support.google.com/accessibility/android/answer/6006564.
[7] Androidguide. 2019. Android Accessibility Developer Guidelines. https://developer.android.com/guide/topics/ui/accessibility
[8] androidmonkey. 2019. Application Exerciser Monkey:Android Developers. https://developer.android.com/studio/test/monkey.html
[9] Androiduse. 2019. Android User worldwide. https://stuff.mit.edu/afs/sipb/project/android/docs/about/index.html
[10] AndroZoo. 2019. AndroZoo. https://androzoo.uni.lu
[11] AppleAccessibility. 2018. Apple Accessibility - iPhone. Accessed April 12th, 2018. https://www.apple.com/accessibility/iphone/
[12] Appleguide. 2018. Apple Accessibility Developer Guidelines. https://developer.apple.com/accessibility/ios/
[13] AppleScanner. 2019. Apple Accessibility Scanner. https://developer.apple.com/library/content/documentation/Accsssibility/Conceptual/AccessibilityMacOSX/OSXAXTestingApps.html
[14] Young-Min Baek and Doo-Hwan Bae. 2016. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 238–249.
[15] BBC. 2019. Mobile Accessibility Standards and Guidelines v1.0. http://www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile_access.shtml
[16] BlindUserworldwide. 2019. Blind User worldwide. http://www.who.int/mediacentre/factsheets/fs282/en/
[17] Santiago Liñán Christopher Vendome, Diana Solano 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.
[18] Trinidad Domínguez Vila, Elisa Alén González, and Simon Darcy. 2018. Website accessibility in the tourism industry: an analysis of official national tourism organization websites around the world. *Disability and rehabilitation* 40, 24 (2018), 2895–2906.
[19] Werner Dubitzky, Olaf Wolkenhauer, Hiroki Yokota, and Kwang-Hyun Cho. 2013. *Encyclopedia of systems biology*. Springer Publishing Company, Incorporated.
[20] Marcelo Medeiros Eler, José Miguel Rojas, Yan Ge, and Gordon Fraser. 2018. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 116–126.
[21] espresso. 2019. Espresso : Android Developers. https://developer.android.com/training/testing/espresso
[22] forbes. 2019. forbes. https://www.forbes.com/top-digital-companies/list/
[23] Google. 2018. google/Accessibility-Test-Framework-for-Android. https://github.com/google/Accessibility-Test-Framework-for-Android
[24] Khronos Group. 2019. The Industry's Foundation for High Performance Graphics. https://www.opengl.org/
[25] Shuai Hao, Bin Liu, Suman Nath, William GJ Halfond, and Ramesh Govindan. 2014. PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*. ACM, 204–217.
[26] IBMAccessibility. 2018. IBM Accessibility Checklist for 7.0. Retrieved March 10, 2018. https://www.ibm.com/able/guidelines/ci162/accessibility_checklist.html.
[27] Susanne Iwarsson and Agnetha Ståhl. 2003. Accessibility, usability and universal design—positioning and definition of concepts describing person-environment relationships. *Disability and rehabilitation* 25, 2 (2003), 57–66.
[28] Mark Kasunic. 2005. *Designing an effective survey*. Technical Report. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.
[29] Royce Kimmons. 2017. Open to all? Nationwide evaluation of high-priority web accessibility considerations among higher education websites. *Journal of Computing in Higher Education* 29, 3 (2017), 434–450.

[30] Elmar Krainz, Klaus Miesenberger, and Johannes Feiner. 2018. Can We Improve App Accessibility with Advanced Development Methods?. In *International Conference on Computers Helping People with Special Needs*. Springer, 64–70.
[31] Lint. 2019. Improve your code with lint checks. https://developer.android.com/studio/write/lint?hl=en
[32] Flávio Medeiros, Márcio Ribeiro, Rohit Gheyi, Sven Apel, Christian Kästner, Bruno Ferreira, Luiz Carvalho, and Baldoino Fonseca. 2017. Discipline matters: Refactoring of preprocessor directives in the# ifdef hell. *IEEE Transactions on Software Engineering* 44, 5 (2017), 453–469.
[33] Lauren R Milne, Cynthia L Bennett, and Richard E Ladner. 2014. The accessibility of mobile health sensors for blind users. (2014).
[34] nltk. 2019. Natural Language Toolkit¶. https://www.nltk.org/
[35] Kyudong Park, Taedong Goh, and Hyo-Jeong So. 2014. Toward accessible mobile application design: developing mobile application accessibility guidelines for people with visual impairment. In *Proceedings of HCI Korea*. Hanbit Media, Inc., 31–38.
[36] Leonardo Passos, Rodrigo Queiroz, Mukelabai Mukelabai, Thorsten Berger, Sven Apel, Krzysztof Czarnecki, and Jesus Padilla. 2018. A study of feature scattering in the linux kernel. *IEEE Transactions on Software Engineering* (2018).
[37] Neha Patil, Dhananjay Bhole, and Prasanna Shete. 2016. Enhanced UI Automator Viewer with improved Android accessibility evaluation features. In *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*. IEEE, 977–983.
[38] qualtrics. 2019. qualtrics. https://www.qualtrics.com/. Accessed: 2019-08-17.
[39] Robolectric. 2019. robolectric/robolectric. https://github.com/robolectric/robolectric
[40] 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*. ACM, 2–11.
[41] 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*. ACM, 119–130.
[42] scanner. 2019. Accessibility Scanner - Apps on Google Play. https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en_US
[43] 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.
[44] Android Accessibility Study. 2019. Android Accessibility Study. https://github.com/Abdulaziz89/accessibility_eval/
[45] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. An empirical comparison of model validation techniques for defect prediction models. *IEEE Transactions on Software Engineering* 43, 1 (2016), 1–18.
[46] Jason Taylor. 2018. 2018 ADA Web Accessibility Lawsuit Recap Report [BLOG]. https://blog.usablenet.com/2018-ada-web-accessibility-lawsuit-recap-report
[47] Unity Technologies. 2019. Unity. https://unity.com/
[48] w3c. 2019. w3c. https://www.w3.org/WAI/standards-guidelines/
[49] 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.
[50] WCAG. 2019. Web Content Accessibility Guidelines (WCAG) Overview. https://www.w3.org/WAI/standards-guidelines/wcag/
[51] WebAIM. 2018. Screen Reader User Survey 7 Results. Retrieved May 10, 2018. https://webaim.org/projects/screenreadersurvey7/
[52] Brian Wentz, Dung Pham, Erin Feaser, Dylan Smith, James Smith, and Allison Wilson. 2018. Documenting the accessibility of 100 US bank and finance websites. *Universal Access in the Information Society* (2018), 1–10.
[53] who. 2019. World Health Organization. (2011). World Report on Disability. http://www.who.int/disabilities/world_report/2011/report/en/
[54] Shunguo Yan and PG Ramachandran. 2019. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing (TACCESS)* 12, 1 (2019), 3.