Can everyone use my app? An Empirical Study on Accessibility in Android Apps

Christopher Vendome
Miami University
Oxford, OH
vendomcg@miamioh.edu

Diana Solano
Universidad de los Andes
Bogotá, Colombia
dc.solano10@uniandes.edu.co

Santiago Linán
Universidad de los Andes
Bogotá, Colombia
s.linan10@uniandes.edu.co

Mario Linares-Vásquez
Universidad de los Andes
Bogotá, Colombia
m.linaresv@uniandes.edu.co

***Abstract*—Universal design principles aim to improve accessibility by ensuring product designs consider all users, including those with certain disabilities (*e.g.,* visual impairments). In the case of mobile apps, accessibility is mostly provided by existing features in mobile devices, like TalkBack on Android that reads information to users. However, it is not clear to what extent developers actually implement universal design principles or utilize these technologies to support accessibility of their applications. By performing a mining-based pilot study, we observed developers seldom use Accessibility APIs and there is a limited usage of assistive descriptions. Then, we focused on understanding the perspective of developers through an investigation of posts from StackOverflow. We identified the aspects of accessibility that developers implemented as well as experienced difficulty (or lack of understanding). We performed a formal open-coding of 366 discussions threads with multi-author agreement to create a taxonomy regarding the aspects discussed by developers with respect to accessibility in Android. From the qualitative analysis, we distilled lessons to guide further research and actions in aiding developers with supporting users that require assistive features.**

***Index Terms*—Empirical Studies, Mobile Accessibility, Universal Design**

I. Introduction

Universal Design broadly focuses on ensuring the accessibility of products, services, buildings, places, etc. Its foundations are in a set of seven principles: equitable use, flexible in use, simple and intuitive use, perceptible information, tolerance of error, low physical effort, size and space for approach and use [1], [2]. These principles aim to avoid disenfranchising *any individual*, despite the potential of having a disability (*e.g.,* color blindness). To emphasize the importance of accessibility, governments have certain laws related to accessibility, such the (i) Americans with Disability Act (ADA) [3] and (ii) Section 508 of the U.S. rehabilitation act [4]. The latter law specifically requires information and communication technology of federal agencies to meet certain standards.

According to the "World Report on Disability" (2011) of the World Health Organization (WHO) [5], "About 15% of the world's population lives with some form of disability [...]." The disability statistics portal maintained by Cornell University reports that "12.9 percent of females of all ages and 12.7 percent of males of all ages in the US reported a disability" [6]. So, it raises a question regarding whether mobile developers are considering accessibility aspects in their apps.

In mobile app development, there is a large body of work considering energy consumption [7]–[11], performance [12]–[15], security/malware detection [16], and automated testing [17]–[20]. However, there is very limited work related to accessibility in Android apps. It is not clear to what extent developers implement universal design principles or utilize accessibility features in their apps. Additionally, there are few works aimed at investigating the challenges that developers face with respect to accessibility (*e.g.,* using some accessibility API or using existing support like TalkBack).

In this paper, we investigate the accessibility aspects discussed by developers and to what extent certain accessibility features are implemented in Android apps. With this knowledge, the research community can create tools to better support mobile developers with accessibility as well as encourage more developers to consider universal design with automated support. To this goal, we first conducted a preliminary mining-based study on Android apps hosted on GitHub to investigate the extent to which open source developers use accessibility APIs and use assistive content descriptions (utilized by screen reading tools). We analyzed 13K+ Android apps and found the vast majority (over 97%) do not use any accessibility APIs. Additionally, only approximately half of the apps have content descriptions for all of their elements. The lack of accessibility APIs suggests a potential lack of consideration or knowledge related to accessibility, and the lack of assistive content descriptions limits the effectiveness of existing support features providing screen reading for visually impaired users.

After the preliminary study, we mined StackOverflow, identifying 810 discussions related to universal design and accessibility; then, we performed a formal open-coding on a statistically significant sample of 366 discussions. From this coding, we present a taxonomy of 58 categories related to aspects discussed by developers. Surprisingly, about 36% of the sample of analyzed questions relates to questions about using accessibility APIs for purposes *not related* to accessibility. However, support for visually impaired users is the most discussed aspect with about 43% of the sample.

To the best of our knowledge, this work is the first to investigate accessibility from the perspective of features implemented in Android open source apps and aspects discussed by Android developers. The limited usage of accessibility APIs demonstrates a great potential to aid developers with enhancing the accessibility of their apps. Additionally, the taxonomy demonstrates areas that developers would benefit from support

2576-3148/19/$31.00 ©2019 IEEE
DOI 10.1109/ICSME.2019.00014

when trying to implement universal design principles.

II. Accessibility in Mobile Apps

Universal Design (UD) is a concept created by the architect and designer Ronald L. Mace in the decade of 1980. In its origins, it was related to architecture, housing and urban development; nevertheless, it evolved to the idea of designing aesthetic and usable products, regardless the condition or abilities of its users. [21]. The underlying idea of "design for all' or "design for everyone" was later translated by The Center for Universal Design into 7 principles [1], [2]:

- "**Equitable use**: the design is useful and marketable to people with diverse abilities;
- **Flexibility in use**: the design accommodates a wide range of individual preferences and abilities;
- **Simple and intuitive use**: use of the design is easy to understand, regardless of the user's experience, knowledge, language skills, or current concentration level;
- **Perceptible information**: the design communicates necessary information effectively to the user, regardless of ambient conditions or the user's sensory abilities.
- **Tolerance for error**: the design minimizes hazards and the adverse consequences of accidental or unintended actions;
- **Low physical effort**: the design can be used efficiently and comfortably and with a minimum of fatigue;
- **Size and space for approach and use**: appropriate size and space is provided for approach, reach, manipulation, and use regardless of user's body size, posture, or mobility."

These principles became the basis of some of the guidelines established for web and mobile software design with the purpose of developing accessible apps. Accessibility is a non-functional attribute in software defined as *"The quality of being easily reached, entered, or used by people who have a disability"* [2]. This concept can be described as the overlap between a person's functional capacity and the design of a physical environment [2]. Interactions between people and these environments result in norms and standards that serve as drivers and guidelines for designing services and products for different populations such as kids, people with disabilities, older adults, among others.

Having accessibility in mind, mobile apps have intrinsic challenges to overcome, because they work on small touch devices in a large variety of platforms and versions of operating systems; For example, touch screens do not provide mechanical feedback like physical keyboards would. Consequently, built-in software feedback, tailored input methods, and screen reading capabilities are necessary to enable equitable use, flexibility in use and perceptible information for the impaired population.

Mobile devices include some accessibility features, including screen reading capabilities as in the case of `TalkBack` for Android and `VoiceOver` for iOS. `TalkBack` gives spoken feedback and notifications to users; it is completely integrated with Android since version 3.2, and includes a series of features whose objective is to improve accessibility throughout the operating system and the applications that comply with the Android's specification. A main feature of `TalkBack` is *Explore By Touch* in which users can interact with the screen dragging their fingers and double-tapping, while `TalkBack` announces headers, labels, icons and other assistive content defined by developers [22]. In addition to `TalkBack`, Android provides many others features to improve accessibility to a broader set of users (*e.g.,* hear impaired, speech impaired) such as changing display and font size, increasing contrast, screen magnification, captions in video, among others.

Even though platforms include accessibility services/features to support universal design principles, developers must comply with certain practices in their applications (*e.g.,* set content description to the images, design TextViews and labels to work with different font sizes). However, as of today, few works have been devoted to identify in large-scale studies whether developers are following those practices.

A. Guidelines

The UD Principles have been translated into guidelines specifically focused on mobile applications. For example, Hellman proposed a list of categories which include: navigation and workflow, errors management, search and queries, time, text and language, voice and sound, graphics, figures and numbers and help information [23]. More specific guidelines have been proposed, regarding this by Wentzel *et al.* [24], who defined specific principles for devices that should work unobtrusively and hands-free (*i.e.,* wearables). In addition, other studies have focused on creating guidelines for specific populations such as Ruzic *et al.* [25] and Markus *et al.* [26] that designed guidelines for the aging population.

In order to evaluate if UD principles are being incorporated into mobile devices, Tomberg *et al.* [27] manually reviewed existing lists of accessibility evaluation criteria focused in web, human computer interaction, mobile, and wearables. Tomberg *et al.* made a compendium of the most important guidelines (*e.g.,* BBC Mobile Accessibility Guidelines [28], W3C Accessibility Guidelines [29]). Other approximations to evaluate the UD principles attempt to quantify the accessibility of a design, *e.g.,* Kim *et al.* [30] proposed an index to evaluate mobile phones hardware, based mainly in ergonomics.

Other guidelines have been proposed but for accessibility testing. Eler *et al.* proposed an approach to generate automated tests for accessibility [31]. In particular, they consider: (i) speakable text, (ii) touch size area, (iii) contrast ratio, (iv) duplicate clickable bounds, and (v) clickable span.

B. Previous Empirical Studies

Other research efforts have focused on empirical studies focused on different accessibility aspects. Kocieliński *et al.* investigated the existing accessibility features with virtual QWERTY keyboards on mobile phones and compared it against using an integrated Braille notetaker (existing tool for visually impaired) [32]. The results demonstrated that integration of the Braille notetaker greatly increased the time to complete tasks and they claim existing support is not sufficient to assist the visually impaired. Zhong *et al.* proposed augmenting the touch feature on Android phones to facilitate less precision to a targeted location and assisting with gestures that may not be suitable for individuals with tremors (*e.g.,* double-tapping the same location) [33].

Other examples are Mehta *et al.*, who conducted a study with 12 blind users to investigate the accessibility of date-pickers and to understand how date-pickers could be improved to support blind users [34]; and, the study by Xie *et al.*, which focused on providing support for GUIs responsiveness when connecting smartphones to external displays [35].

Researchers have also explored apps' accessibility in various contexts. Coelho *et al.* manually evaluated 4 government mobile applications using W3C Accessibility Guidelines [29]. They conclude that accessibility problems are extensive in these cases [36]. Walker *et al.* evaluated weather apps by conducting a survey with blind and sighted smartphone users; the study shows that, in general, the apps were not developed to be universally accessible [37]. Al-Subaihin *et al.* discovered that the functionality of `TalkBack` and `VoiceOver` with mobile web apps could be as good as with native applications if structural HTML elements are correctly used [38]. Krainz *et al.* propose to improve accessibility in mobile platforms by changing the app development process; their main clonclusion is that a model-driven approach with automated code generation could avoid accessibility problems [39].

Accessibility for users with special needs has been also studied in mobile apps. Araújo *et al.* provided a manual test to evaluate mobile audio games; this test was developed in order to check that audio games meet the need of visual impaired users [40]. In the case of older adults, Díaz-Bossini *et al.* [41], [42] have assessed mobile accessibility and proposed guidelines based on studies with older users about their perceptions of mobile applications and their specific requirements.

The aforementioned studies have been conducted in a small scale, targeting a specific population of apps and users, and mostly from the users' perspective. Note that none of the studies has focused on developers and apps at a large scale.

C. Tools for Assessing Accessibility

There are three major official tools for Android: Accessibility Scanner [43], Lint [44], and Node Tree Debugging [45].

Accessibility Scanner (AS) is an Android tool that runs on devices and scans the current snapshot of any app with the purpose of giving suggestions for improving its accessibility. It is based on dynamic analysis, and requires the app under analysis to be installed on a device and be in foreground. AS scans the app and shows accessibility recommendations for each component in the GUI (if any). The suggestions given by the AS are mostly related to increasing the size of the components or improving the contrast ratio of the colors used for the background and fonts.

Lint is an static code analyzer that runs as part of the SDK but also integrated with the Android Studio IDE. It reports micro-optimization opportunities, potential code errors, and other issues related to different quality attributes. In terms of Accessibility, Lint detects the following issues: missing content descriptions and missing accessibility labels declared directly in XML layout files. One drawback with Lint is that it requires to have access to the source code and the app should use gradle as the build automation tool.

Node Tree Debugging is more a tool for testing Android apps. This tool is able to describe how an AccessibilityService in the app interprets the app's UI elements. In order to use Node Tree Debugging, it is necessary to enable an AccessibilityService like `Talkback`; then, from the manual interaction with an app, the Node Tree is printed in the Logcat. By using the Node Tree Debugging tool, developers can identify the focusable elements and their assistive descriptions.

Finally, there are also unofficial tools like the "Enhanced UI Automator Viewer" described in Patil *et al.* [46], which extends UI Automator to capture snapshots of an app under analysis and "audit" the snapshots looking for accessibility issues. Specifically, the tool detects unlabeled UI elements and color contrast. However, analyzing an app with the tool requires it to be installed in a virtual or physical device.

III. Motivational Mining Study

While the main focus and contribution of this paper is a qualitative study on the developer perspective on accessibility (*e.g.,* their difficulty supporting accessibility, the features that they are implementing, etc.), we first performed a pilot study to understand whether developers were using certain assistive features, namely Accessibility APIs and assistive content descriptions for GUI components. It is important to note that this motivational study aims to provide an initial empirical insight towards developers supporting accessibility by analyzing the source code of apps using light-weight methods; however, a more comprehensive evolutionary study would be needed to truly understand the extent to which developers are implementing accessibility support in Android apps.

For this motivational study, we randomly selected 13,817 Android apps from GitHub that had at least one follower, star, or fork to avoid abandoned projects, and we ensure the project itself was not a fork to avoid duplicates in our dataset. The projects were automatically identified by the presence of an *AndroidManifest* file and we removed apps without at least one Android activity. The revision history of these apps ranged from a single commit to 66k+ commits. To automate our analysis, we built a lightweight tool to statically analyze the repositories of these apps to identify (i) accessibility APIs used in the apps source code, and (ii) presence/lack of assistive content descriptions in the GUI components. For the accessibility APIs, our tool detected whether the `android.accessibilityservice` and `android.view.accessibility` packages are imported in the app's source code. Note that we did not rely on the existing Android tools because (i) they are not tailored for the case of automated large scale analysis (*e.g.,* Accesibility Scanner and Node Tree Debugging), and (ii) in the specific case of Lint, it requires projects with gradle scripts.

For the assistive content descriptions, the tool analyzed (statically) the layout files to identify the UI components and then checks if the components have a non-empty `contentDescription` attribute (for `ImageView` and `ImageButton` components) or a non-empty hint attribute (for `EditText` components) in the XML file. Additionally, the tool analyzed the source code to identify invocations to `setContentDescription()` and `setHint()` methods to identify dynamic labeling of elements, because assistive content can be also set programmatically. It is important to note that this content description attribute is utilized by screen readers like `TalkBack` (integrated in Android by Google) to provide support for visually impaired users; screen readers in the mobile operating systems work by reading the text associated with elements on the screen as a user navigates the screen by touch – typically referred to as *Explore by Touch* – so that a user can gain an understanding of the screen.

In terms of the accessibility APIs usage, we observe that only 288 (2.08%) out of 13,817 Android apps import at least one accessibility API. Looking into the accessibility APIs, we observe that 252 apps imported a single accessibility API, 14 apps imported two accessibility APIs, and 22 apps imported 3 accessibility APIs; in total, the APIs were imported 346 times. The most prevalent APIs were *android.view.accessibility.AccessibilityEvent* (286 imports), *android.accessibilityservice.AccessibilityServiceInfo* (33 imports), and *android.accessibilityservice.AccessibilityService* (24 imports). The remaining three import sentences were for entire packages (imported using the '*' wildcard). It is worth noting that Android accessibility APIs can be used also with non-accessibility purposes such as creating key-loggers or notifications interceptors. However, our purpose with the motivational study was not to identify the specific usages of the Accessibility APIs.

We observed that 6,920 Android apps (50.08%) have assistive content for all their GUI components. Conversely, we observe that 6,390 apps (46.25%) have *at least half of the elements with empty content descriptions* (of these 6,390 Android apps, 5,107 of them had *all* of the content descriptions empty). Thus, approximately half of the Android apps do completely include assistive content that can be used by screen readers. Note that we were not interested on assessing the quality of the assistive content; also we used a lightweight tool because we were interesting on having a preliminary overview of the prevalence of accessibility APIs and assistive content in Android apps.

Summary: The motivational study found that approximately half of the projects provided assistive content descriptions, but seldom utilized APIs for accessibility. This observation suggests that there is a consideration for providing accessibility support at least in terms of including assistive content for GUI components. However, the developers tend rely on mechanisms that facilitate this support as opposed to implementing the features through APIs, but there were still a substantial proportion of apps that did not implement assistive contents at all. Future work should be devoted to identify specific usages of the accessibility APIs and reasons for the lack of a more prevalent implementation of accessibility features in Android open source apps.

IV. Empirical Study

The *goal* of this study is to identify the accessibility practices followed by developers in Android apps, and in particular the ones oriented to allow app usage by users with disabilities. We follow a mining-based study in which we analyzed developers' discussions at Stack Overflow. The *perspective* we followed is the one of researchers interested in promoting (i) universal design practices in mobile apps, and (ii) envisioning automated tools for implementing accessibility features in Android apps. In particular, we investigated the following research question:

RQ$_1$: *What accessibility aspects are discussed by developers in Stack Overflow?*

With this RQ, we aim to understand the types of accessibility features that developers are implementing and the issues that they face with respect to accessibility. In order to better support developers with accessibility, we believe it is important to understand the difficulties that they encounter.

A. Data Collection

We identified posts from StackOverflow related to accessibility features and concepts. First, we collected candidate discussions by mining questions and answers with the following keywords: *accessibilit, blind, braille, brailleback, deaf, deaf people, design for everybody, disabilit, hearing loss, impair, medical, on-screen text, real-time text, screen reader, select to speak, switch access, talkback, tremor, universal design, vision impairment, visually impaired, voice access, voice over*. Note that we performed a substrings matching such that keywords like *disabilit* could match with disability or disabilities.

The list of keywords was selected from universal design and accessibility guidelines for mobile apps, having in mind accessibility features (*e.g.,* TalkBack) and user disabilities (*e.g.,* hearing loss). In addition, we focused only on Android-related posts, *i.e.,* we queried questions and answers from StackOverflow (SO) with any tag including the term *android*.

To collect the SO posts (*i.e.,* questions and answers), we used the stack exchange data explorer tool (https://data.stackexchange.com/stackoverflow/query/new), and we limited the query to retrieve posts with creation time no further than November 2018. At the end, we collected a set of 1,442 discussions. Note that a discussion includes a question and the list of answers provided for that given question. It is important to note also that each of the authors has experience in mobile app development and knowledgeable of accessibility features.

Some of the keywords used in the query are prone to false positives (*e.g.,* the term *accessibility* is also used when referring to connectivity like network accessibility) and code entities accessibility (*i.e.,* public, private, etc.). Therefore, one of the authors went through the 1,442 discussions and discarded the ones not related to accessibility features/concepts in Android. After filtering the false positives, the number of discussions was reduced to 810.

Finally, we manually categorized a statistically significant sample (with 99% of confidence and margin error of 5%) of the filtered discussions by following an open-coding inspired process, composed of 366 discussions. The sample was selected randomly from the 810 filtered discussions. Afterwards, the 366 discussions in the sample were distributed among the four authors in such a way that each discussion was analyzed by two "coders". Before starting the analysis, each coder was instructed to read and assign multiple "codes/tags" to each of the assigned discussions with the purpose of identifying (i) universal design-related aspects mentioned in the discussion, (ii) issues experienced by practitioners, (iii) how the practitioners are using the Android accessibility APIs (or other related APIs), and (iv) accessibility-related features implemented by practitioners.

Note that it was a multi-tagging process, which means that four free-form tags were assigned to a discussion (one for each of the aforementioned aspects). Also, the tags *False Positive* and *Unclear* were available in case the coder considered the discussion as a false positive or she/he was not able to find an appropriate tag for the artifact, respectively. The tagging was assisted by an online tool in which every time a new tag was added to the list, it was available to the four coders. The tagging was done in iterations to allow clarification and merging of tags. A first iteration was done with 50 Stack Overflow discussions (each coder), a second one with 60, and a last one with 73 artifacts. In total, each coder assigned tags to a total of 183 discussions. After each iteration, the authors discussed regarding the generated tags looking for merging and expressiveness improvement opportunities.

After the three open-coding iterations, we obtained 56 (15.30%) discussions without conflict (*i.e.,* the four tags assigned by the coders analyzing the discussions were the same); in 89 posts (24.32%), there were conflicts in three tags; in 114 cases (31.15%), there were conflicts in two tags; in 79 cases (21.58%), there was conflict only in one tag; and in 28 cases (7.56%), the coders completely disagree (*i.e.,* no tags in common). To solve the conflicts in the 311 posts with at least one tag disagreement, given a post P, a third author (not previously involved in the coding of P) was assigned to solve the conflict, *i.e.,* the third coder decided the four tags while having access to the tags provided by the other two coders. Note that the names of the original coders were anonymized.

B. Analysis Method

To answer **RQ**$_1$, we built a taxonomy of accessibility aspects discussed by developers in Stack Overflow. The taxonomy was built by the four authors based on the coding process, and through an open discussion with the purpose of organizing accessibility aspects in a hierarchy. Iteratively, the codes/tags were organized in higher concepts until all the codes were exhausted. The taxonomy is the set of concepts organized in a hierarchical way.

V. Results: What accessibility aspects are discussed by developers in Stack Overflow?

As a result of the open coding process, we obtained a total of 149 tags, and 30 discussions in which there was no agreement regarding whether they are related/relevant to accessibility. Thus, the taxonomy was built with 336 discussions (after removing the aforementioned 30 issues) that are organized into 58 categories. Figure 1 depicts the taxonomy and its categories. In the following, we describe the 6 top-level categories and their sub-categories; because of space limitations, we include qualitative examples only for some of the sub-categories. When referring to categories and subcategories in the taxonomy, we will use italic font (*e.g., accessibility warnings*).

A. Support for visually impaired people (159 discussions)

This is the top-1 discussed aspect, in particular because of questions (114) related to *sound-based feedback* features that describe what is displayed in the app's GUI to the user. Android supports visually impaired users with the `TalkBack` feature for screen reading, and via custom features for using sounds and vibration as a response to touch-based exploration; those custom features can be implemented by developers with the `Accessibility` APIs .

Screen reading is implemented in Android apps by specifying, in layout files or via `Accessibility` APIs, the assistive content that is going to be spoken to the user when she explores the GUI using touch events while `TalkBack` is active. Based on our analysis, we found that Android developers look for help in StackOverflow with understanding how to use `TalkBack` and how to customize the assistive content to be reproduced. For example, the **SO question `25244457`** asks about setting up `TalkBack` for reading the time-of-day:

> "Somethings I am struggling with: Times are spoken as "seven point one five pm" for 7.15pm; "pm" is ok but "am" comes out as the word "am" as in "I am"; dash is spoken as minus when used to separate words. How can I fix this?"

There are other examples of questions related to *custom content/action/gestures*. While **SO question `53547488`** describes the need for customizing the way abbreviations and initialisms are pronounced, **SO question `17181511`** asks about how numbers should be pronounced by a screen reader. In the same line of thought, **SO question `39455917`** discusses changing the defaults instructions provided by `TalkBack`:

> "I use the content description to set the message Talkback speaks when the view gets focus. Currently it says my content description right after getting a focus, and after a short pause says: 'Double tap to activate, double tap and hold for long press'. I want to change this message into something like Double tap to 'action 1', double tap and hold for 'action 2'. Is there a way to do so?"

We also found questions discussing the quality of screen reading services in Android. The **SO question `14080636`** discuss the quality of `TalkBack` as compared to `VoiceOver` in iOS. **SO question `15181652`** describes that `TalkBack` is disabling custom gestures, and **SO question `25867034`** reports that the `Text-To-Speech` API generates Application-Not-Responding errors when using long texts. Concerning non-functional aspects, in **SO question `24462690`**, the developer

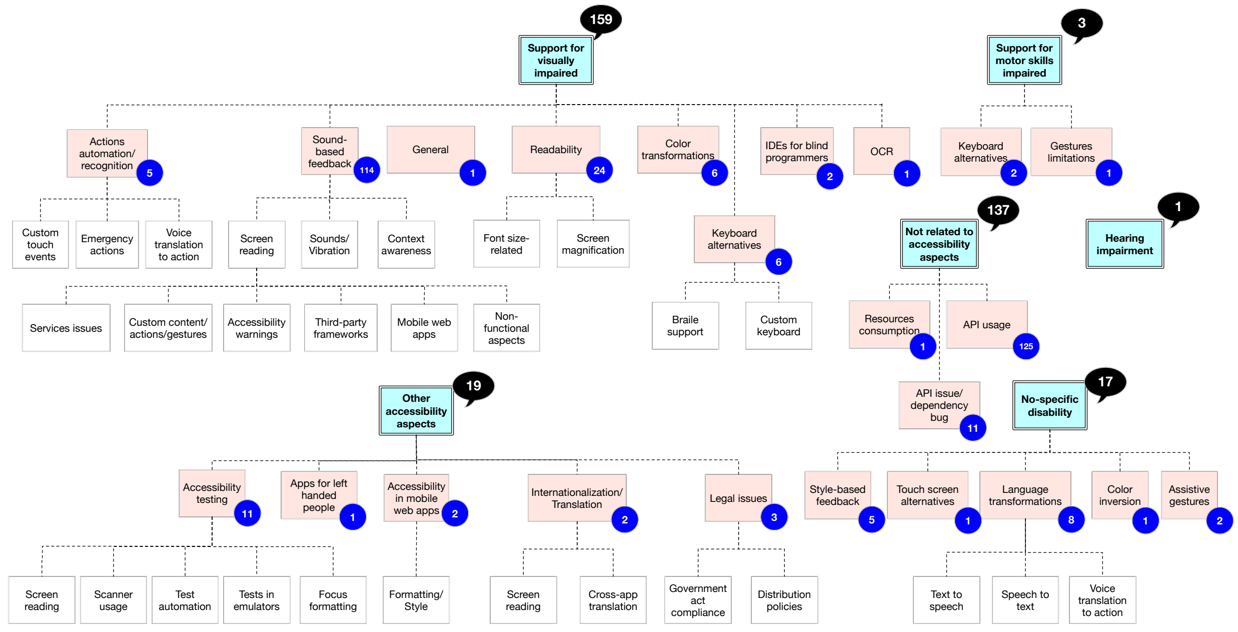

Fig. 1. Taxonomy of accessibility-related aspects discussed by Android developers in Stack Overflow

is concerned about the performance impact of setting long assistive texts.

Screen reading-related issues are also experienced by developers, when using *third party frameworks* for hybrid or *mobile web apps*. For example, **SO question `35230632`** illustrates that `TalkBack` requires customization for screen reading specific HTML tags.

There are other features discussed by developers, for sound-based feedback that are not supported directly by the `Accessibility` APIs. For instance, **SO question `7384101`** is an example of *context awareness* for blind users; the developer is looking for help when implementing an app that requires detecting the presence of a bar code (using camera) and then notify the user via vibration or synthesized voice.

The analyzed posts also show that some developers are lacking knowledge on *accessibility warnings* generated by Android when assistive contents are not specified, as in the case of **SO questions `9363171, 47107105, 8500544`**. In those questions, the developers request for help to disable the warnings and look for explanations of the warnings purpose.

Another accessibility aspect of interest for practitioners developing apps for visually impaired users is *readability* (24 discussions). This aspect includes discussions related to ensuring that the text is readable via responsive *fonts* when (i) the large text feature is enabled in devices, (ii) the app is installed on different device sizes, and (iii) enabling *screen magnification*.

A representative example of *font size-related* discussions is **SO question `16383638`** in which the developer describes that despite having support for different screen sizes (small, normal, large, extra large), the layout exceeds the screen size when an app user selects the "large text" accessibility setting. Another similar example is **SO question `19817086`** in which the developer asks for "a way to fix text sizes independent from device's settings". This issue is also exhibited in the case of mobile web apps as illustrated in **SO questions `24911220, 18274504, 42955105`**.

Concerning *screen magnification*, developers look for advice about enabling screen zoom in mobile web and native apps for accessibility purposes. In the **SO question `16815416`**, the developer asks for advice with his native app:

> "I would like to create an Android Accessibility Application/Service. This Accessibility app would be able to magnify any screen image produced by any application resident on the android device"

In addition, in **SO question `18274504`**, the asker requests about enabling zooming in a mobile web app:

> "That's right, I want to enable zooming rather than disabling it. I have created a responsive website and all is looking good. However for some reason you can't pinch-zoom in Chrome on a Nexus 7 (running Android 4.2.2)."

Conversely, developers also look for help in StackOverflow to solve issues introduced in their apps when screen magnification is enabled or when screen resolution is reduced (*e.g.,* **SO questions `16383638, 43698579`**).

In the top-3 position of aspects related to visual impairments, with 6 questions each, we found *color transformations* for color-blind users, and *keyboard alternatives* to the default Android keyboard. Color-blind users are unable to distinguish certain colors, and in our sample of StackOverflow discussions, we found developers asking for help in terms of color transformations that should be used in mobile apps for color-blind users. For instance, the author of **SO question `24595021`** is developing a game for color-blind users:

> “Ideally, I would like a method that just takes any color, and converts it to the color a color blind person would need to see to perceive the original color”

The discussion in **SO question `11461336`** illustrates that for color-blind users, information in GUIs should be transmitted in different ways:

> “The problem is you need to transmit information by using color but the screen (or color-blind people’s eyes) remove some information from the color so the users will be confused. The solution is to make sure the information transmitted by color is not affected by the screens or eyes, and you can also use other media rather than color.”

There are also some recommendations regarding color palettes and other options (**SO question `11461336`**):

1) “Color-blindness or grey-scale screen doesn’t remove all information in the color. For example, if color is described as HSL (Hue/Saturation/Light), grey-scale screen removes Hue and Saturation but keeps Light. If you use 5 different colors, as long as they have 5 distinctive Light values (e.g. 0.1, 0.3, 0.5, 0.7, 0.9) users won’t have a problem recognize them in grey-scale screen.”
2) “You can also use other media to transmit the information transmitted by color. For example, Adium for Mac uses both shape and color to indicate contact status. Online user is green rectangle, away is yellow triangle, and busy is red circle. This is color-blindness friendly because users can understand contact status by identifying the shape anyway.”

Visually impaired users can also be assisted by *alternatives to the default keyboard* displayed in the screen of Android apps. One example of these alternatives is the braille keyboard, which is the discussion topic in **SO questions `35165206,43734431`**:

> “I’m working on a project for blind people, there’re a lot of troubles I need to fix if the user activates TalkBack on his phone. I’m creating a soft keyboard for blind people, the blind will tap with single finger on the circles "Braille Cell Dots" to generate a Braille code, then he types the character/number/symbols he wants as they presented in Braille language.”

Other alternatives, different than braille keyboards, are external keyboards and custom on-display keyboards (*e.g.,* **SO questions `4910035,9880049,26345497`**).

The last set of aspects that we found that are related to visual impairment is the *automation of actions* (5), and *IDEs for blind programmers* (2). The former category includes questions about customizing touch events, automating emergency calls, and triggering actions in the device via voice commands. The latter category includes questions about lack of accessibility in the Android emulator for a blind programmer (*e.g.,* **SO question `3382130`**), and customization of color palettes in Android Studio and IntelliJ for a color-blind programmer (*e.g.,* **SO question `37268963`**).

B. Support for motor skills impaired people (3 discussions)

Fine motor skills are fundamental to interact with GUIs in mobile devices via touch gestures (*e.g.,* drag to scroll, spread to zoom in, pinch to zoom out). Limitations in motor skills do not allow people to execute specific gestures, as reported by **SO question `11443820`**, which contains an explanation on how to overlay a button on an Android application as a solution for users that are unable to scroll the screen looking for buttons. Another issue associated with motor skills is the interaction with the Android keyboard. **SO question `47099396`** relates to this and it discussed how to set as default a specific virtual keyboard; however, a developer raised concern about the accessibility issues it could create: "*People with physical disabilities like Parkinson’s, blindness, etc frequently use specialized keyboards. You should not take that ability from them.*" **SO question `30942775`** discusses how to use Accessibility Services to automate text inputs and gestures to overcome fine motor skills limitations.

C. Hearing impairment (1 discussion)

SO question `25800377` focuses on real-time call transcription in Android using the `SpeechRecognizer API`. The main point of the discussion is to describe the best way to transcribe phone calls for a long period of time in Android. Answers suggest moving the speech recognition service to a server due to performance and API restrictions.

D. No-specific disability (17 discussions)

We found accessibility related questions (17) that were not oriented to a specific disability, but used Accessibility APIs or Accessibility services. In this category, the most common questions relate to *language transformations*: text-to-speech (TTS), speech-to-text (STT) and voice translation to action. Of these questions, two corresponded to STT issues, four to automation of actions from voice commands and two to usage of the TTS API.

The second most common questions not oriented to a specific disability were *style-based feedback* with 5 questions. These questions were mainly related to the focus state of components in the view and the highlight behavior in components when pressed. The remaining questions in this category were about alternatives to touch screens, transformation of screen colors, and implementation of assistive gestures.

E. Other accessibility aspects (19 discussions)

Concerning *Accessibility testing*, **SO question `26490845`** discusses the question "*How to automate test for Accessibility on Android?*" The author of the discussion wants to create end-to-end accessibility tests that could be automated. In brief, none of the answers gives an easy-to-use solution, and one response redirects the developer to **SO question `22385953`** in which the same topic is discussed. In this case, a user writes a comprehensive answer summarizing an external article that explains the subject. First, he suggests manually testing the application with Accessibility Scanner, turning on and off TalkBack, changing the font size, and checking the project with Android's Lint. For test automation, he suggests writing tests in Espresso (also explored in **SO question `46097973`**) and analyze audio/visual aspects with external commercial tools.

Internationalization/Translation is another relevant accessibility aspect. It is focused on making software adaptable to different languages and regions. **SO question `18516516`** explains in detail how to manage translations of labels and assistive content descriptions.

The third topic in *Other accessibility aspects* is *Accessibility in mobile web applications*. **SO questions `12433062, 10138968`** discuss how to set CSS rules in web views and websites to change the style of the accessibility focus frame that appears on buttons when they are clicked. In both questions, users warn other developers not to disable the focus frame or make it invisible.

A very interesting category discussed are *Legal issues*. In **SO question `51320276`**, the developer asks if using accessibility services for purposes not-related to implementing accessibility features in apps is allowed by the Google Play distribution policies. The accepted answer for the question states that this is not possible, and the app could be removed from the Google Play Store. In fact, **SO question `52059795`** shows an explicit case where an app was removed as a result of violation of this policy. Another question related to legal issues asks how to make an app compliant with government acts of accessibility (**SO question `6900056`**). In general, answers suggest that taking into account Android and iOS accessibility guidelines is enough, and that way, applications could be compliant with the legal standards.

The last topic discussed is *Apps for left handed people*. In **SO question `27206868`**, the developer is trying to detect if the Left Hand Mode or `Force RTL Layout Direction` is enabled in the device settings. This was an old post, and the developer was developing and app for Android 2.3. Note that this version of Android had a very limited support for RTL. However, from Android 4.2, Google added full native support for RTL and left handed users.

F. Aspects not related to accessibility (137 discussions)

The analyzed sample also included discussions related to *Accessibility API usages* that are not-related to any accessibility aspect. The `Accessibility` API allows implementing different automation tasks that can even be executed in background and monitor/modify other apps behavior. For instance, the `Accessibility` API has been widely used in testing tools (*e.g.,* [47], [48]) for automating execution of events in the GUI and for data extraction.

In 125 questions, developers discussed about using `Accessibility` APIs with different purposes such as monitoring other apps, starting/setting up/stoping accessibility services, automating tests and simulating ui-interactions, detecting notifications, copying & pasting, among others. In **SO question `23059868`**, a developer asked *"How can I retrieve the name, message and time for any incoming notification on 'whatsapp'."* The provided answer asserts that it can be done using accessibility services:

> "I was able to do this using Accessibility Service. Using this, you can listen to all notification on the notification bar. I listened to application-specification by adding the package name to the Accessibility Service service info, which in this case was com.whatsapp. I couldn't read the messages, but I get notified whenever a message arrives."

Another interesting example is **SO question `50103093`** in which the user asked for help when implementing copy-paste in an Android app:

> "I have used it to copy and paste my clipboard text to wherever my current text cursor is focused but it is not working? I am kind of newbie to this accessibility service"

The aforementioned examples and the rest of questions that we found in this category (*i.e., API usages*) show the Android `Accessibility` API has been used for purposes other than the ones expected by Google. According to the official developer reference [49], *"Accessibility services should only be used to assist users with disabilities in using Android devices and apps."*

As a response to the improper usage of the API, recent versions of Android include a new usage policy and warnings that are thrown when the accessibility services are not used properly as illustrated in answers to **SO questions `50103093 and 47107105`**:

> "[...] But the problem is that Google some time ago changed policy and now Accessibility Service can be used only for assist users with disabilities."

and

> "This warning comes up because Android wants to remind you to think about the blind or visually impaired people who may be using your app"

The other aspect in this category is *API issues and dependency bugs*. Some of the issues are because of breaking changes in the API as discussed in **SO question `23610393`**:

> "It works and the method onAccessibilityEvent is called everytime a text is changed in any application but, when I run this code in an Android Pie (API 28) only the onServiceConnected one is called. I suppose something has been changed in from API 27 to 28 but I can't find anything on the net."

Other issues are generated because of dependencies of other APIs during building time (*e.g.,* **SO questions `49786779, 26792510`**). Finally, there was one question about the impact on energy consumption of using accessibility services (**SO question `35252210`**).

VI. Discussion

When performing our qualitative study, we observed that support for the visually impaired was the most common category. These discussions heavily relate to (i) screen reading and the abilty to use or customize *TalkBack*, which provides the screen reading capability for Android devices, or (ii) individuals creating specialized keyboards (*e.g.,* to support braille). This observation suggests that at least a subset of developers are interested in implementing accessibility, when they have existing tools that help provide the accessibility features. However, this subset is small compared to population of Android-related questions in StackOverflow; the tag *android* accounted for about 1 million questions at the moment of extracting the questions. The motivational study seems to corroborate this (Section III), since we observed that approximately half of the apps had content descriptions for *all* of their elements.

Conversely, we observed a lack of discussions relating to the other disabilities like motor skill impairments or hearing impairments. We do not have as mature of a support tool integrated into the Android system for these two types of disabilities. In the latter case (*i.e.,* hearing impairment), it may be partly due to the nature of the disability, *i.e.,* a hearing impaired user may be able to interact with more apps since they can read the elements, and sounds can be replaced with vibrations as a feedback mechanism. In our related work search, we did not find empirical studies focused on the aforementioned populations, except for prior work by [33]. Therefore, it would be beneficial to conduct a study with such users to better understand the types of support that they might need when using mobile apps.

Interestingly, in both the motivational and main study, we observe that developer seldom use the Android accessibility APIs in the apps, but it is more frequent that developers provide content descriptions for the GUI elements. In the former case, developers have limited guidance outside of Google's guidelines and Q&A discussion regarding using these APIs. In fact, these APIs can also be utilized to implement features that require to handle certain events from external apps as seen in the SO discussions. Conversely, developers have support for detecting issues related to assistive content with the *Lint* tool, since it provides warnings reporting that content description text is missing. However, we do observe that developers do not necessarily understand the importance of this feature as some discussions on SO indicate that developers do not know the relation between assistive content and some GUI attributes and want to just remove the warnings when displayed by the IDE.

Similarly, we observe that developers have questions regarding the automation of accessibility testing. While *Lint* gives warnings for a particular case of accessibility issues, an integrated (*e.g.,* in the IDE) accessibility-focused automated testing framework that provides both the accessibility issues and conveys the understanding of these issues to developers could benefit the developers. As previously mentioned, developers do not necessarily understand *how* the accessibility warnings from *Lint* impacts the app's usage from a user with a disability. Other tools like the Accessibility Scanner provide other testing capabilities, but it is far from a comprehensive solution for accessibility testing that considers different impairments. This is an opportunity for future research on approaches and tools for automated accessibility testing.

It is worth noting that we do *not* assert that developers do not care about accessibility, but it is something for which potentially they may not be aware, especially when considering the implications for users with certain disabilities. Further studies involving developers and closed-source apps would be beneficial to understand (i) the extent to which they consider accessibility and universal design principles, (ii) the extent to which they know the existence of regulations (*e.g.,* the Section 508 act [4]), and (iii) how to support developers with implementing accessibility features. Similarly, the disparity between the usage of accessibility APIs and use of content descriptions may be impacted by the presence of support for the latter. Additionally, it is not clear how *useful* those content descriptions are to users with visual impairments. In future work, we plan to delve deeper into the semantics of these attributes to understand whether developers are using them *properly* to support accessibility. It is possible that assistive content is not properly internationalized; thus, future work could be also devoted to analyze this aspect and provide tools for automated internationalization of assistive content.

VII. Lessons

Based on the results and discussion, we identified a set of lessons regarding supporting mobile developers. For each lesson, we provide a description based on our work. Additionally, we distilled an actionable finding from each lesson to guide future work in this area.

*1) **Lesson - Developers leverage existing tools to support accessibility**:* Developers frequently asked questions related to *TalkBack* and supporting screen readers. There is also support for developers to identify missing content description attributes provided by *Lint*. It is not surprising that developers inquire regarding accessibility-related warnings as they are alerted by the IDE. Thus, when developers are made aware of accessibility issues, we observe that developers are inclined to learn more about the issue, and they are willing to support accessibility, especially when the burden of the implementation is on external tools (*e.g.,* screen readers).

Actionable Result: Automated approaches to assist developers with identifying accessibility issues should be integrated within the IDE.

*2) **Lesson - Mobile developers lack background and exposure to accessibility features/tools**:* Several of the questions related to screen reading focused on the way text was dictated (*e.g.,* reading a phone number as "five-five-five" as opposed to "five hundred fifty-five") to the user, and developers inquired how to *force* a particular way of reading text. These developers considered how *they* expected to hear the text and not how *visually impaired users* comprehend screen readers. If a developer was made aware of this issue with *Lint*, the message (omitting text about decorative images and utilizing *hints*) that they would receive is:

> "Non-textual widgets like ImageViews and ImageButtons should use the contentDescription attribute to specify a textual description of the widget such

that screen readers and other accessibility tools can adequately describe the user interface..." [50]

This simple example demonstrates a lack of guidance in understanding how users with impairments interact with the mobile applications beyond providing assistive text.

Actionable Result: Developers could benefit from more training on universal design principles and detailed guidance from tools that identify accessibility issues so that they can better understand the *user expectations*.

3) ***Lesson - There is limited support for automatic accessibility testing of diverse disabilities:*** We observed questions related to performing UI automation for accessibility testing. While there are many automated approaches to perform traditional types of testing, the state-of-the-art tools for accessibility testing provide a limited scope of and predominantly supports features focused on visually impaired users (*e.g.,* screen reading, *Explore by Touch*, or text/image size).

Actionable Result: There is an opportunity for the research community to design new robust tools that will automatically test the accessibility of mobile applications.

4) ***Lesson - There is a lack of support for developers with regard to automating the implementation of accessibility features:*** While there is *limited* support to identify accessibility issues, developers need to understand the issue and then implement the fix. As previously mentioned, developers seem inclined to automated support and many questions demonstrated that they have difficulty implementing certain accessibility features. Thus, there is a need for an integrated solution that can assist these developers with supporting accessibility as opposed to only identifying the lack of accessibility.

Actionable Result: (Semi-)automated solutions for implementing and testing accessibility features would greatly benefit developers and mitigate the knowledge gap or lack of experience with respect to accessibility.

5) ***Lesson - Accessibility APIs are utilized for more purposes than just accessibility:*** We observed several questions related to creating Accessibility Services for other purposes than accessibility as these services can serve to get privileged information. For example, developers asked about reading notifications of external apps through Accessibility Services.

Actionable Result: While the Accessibility APIs are designed to provide accessibility support, they can be used to interact with events from external applications that may result in privacy concerns; further investigation and support may be needed to avoid such vulnerabilities.

6) ***Lesson - Mobile app accessibility is not only about screen reading:*** Despite screen reader usage being the most discussed topic in the analyzed questions, a sample of developers are also concerned about other topics related to supporting visually impaired users (*e.g.,* readability for partially visual impaired users and color transformations for color blind users), and other accessibility aspects. However, it is important to note that accessibility related questions are a small sample of the whole population of android-related questions in SO. It suggests that accessibility is a topic for which mobile app developers have limited educational resources and is a topic that needs to be considered more by mobile developers.

Actionable Result: Practitioners, researchers, and educators should devote more effort to identify and promote best practices for designing and implementing accessible apps.

VIII. Threats to Validity

Construct validity: in our motivational study, we performed static analysis to identify the accessibility APIs and usage of content descriptions; and in our qualitative study, we considered posts from SO, but these discussions may not have been complete. It is possible developers considered other principles of universal design that would require dynamic analysis to be identified. We do no assert the *complete lack of accessibility*, but a limited support for accessibility based on the static analysis.

Internal validity: In terms of the qualitative study, we utilized multiple coders for each issue and multiple authors merged similar tags. Additionally, the inter-rater agreement was computed for each aspect that we coded to demonstrate the validity of this analysis.

External validity: Our study relates to discussions from SO regarding Android apps. Other languages, ecosystems, or Q&A websites may have a different focus with respect to accessibility. To limit this threat, we do not generalize our findings to mobile or even all Android apps. Additionally, we do not assert our catalog is complete as it is possible that developers experiencing different difficulties document them through other systems other than SO.

IX. Conclusions

We presented qualitative and quantitative analyses in order to evaluate the awareness of accessibility issues among Android developers. In our motivational study on 13,817 open source Android apps, we found that half of the apps (50.08%) had all the elements labeled with assistive content for screen reading, and the 36.96% of the apps did not have any labeled element. Pertaining to the use of accessibility APIs, we found that only 2.08% of the apps imported at least one accessibility API. We also manually analyzed and tagged 366 StackOverflow discussions related to accessibility. After this process, we built a taxonomy of the aspects discussed by developers, finding out that the most frequent one was *Support for visually impaired people*. From the visually impaired related discussions, sound-based feedback discussions stand out above the others.

Surprisingly, *Not related to accessibility aspects* is the second most frequent aspect and these questions are related to API usages but with purposes other than accessibility. We observed that Accessibility Services are commonly used for non-accessibility aspects, *e.g.,* retrieve notifications from other apps or automate touch interactions in the device. In future work, we aim to understand how developers are using the Accessibility APIs, and how current features cover the needs of impaired users. Additionally, we suggest that the researcher community devote more efforts to investigating the perspectives of both developers and users perspectives towards universal design and accessibility of mobile apps, and how to provide more automated support for accessibility testing.

References

[1] R. L. Mace, B. R. Connell, and M. Jones, "The principles of universal design." [Online]. Available: https://projects.ncsu.edu/design/cud/about_ud/udprinciplestext.htm

[2] S. IWARSSON and A. STÅHL, "Accessibility, usability and universal design—positioning and definition of concepts describing person-environment relationships," *Disability and Rehabilitation*, vol. 25, no. 2, pp. 57–66, 2003, pMID: 12554380. [Online]. Available: https://doi.org/10.1080/dre.25.2.57.66

[3] ADA National Network. Americans with disabilities act (ada). https://adata.org/learn-about-ada.

[4] U. G. S. Administration. Section 508 of the u.s. rehabilitation act. https://www.section508.gov/index.php.

[5] World Health Organization, "World report on disability," Tech. Rep., 2011.

[6] C. University. Disability statistics. http://www.disabilitystatistics.org/.

[7] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, and D. Poshyvanyk, "Mining energy-greedy api usage patterns in android apps: An empirical study," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, ser. MSR 2014. New York, NY, USA: ACM, 2014, pp. 2–11. [Online]. Available: http://doi.acm.org/10.1145/2597073.2597085

[8] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, M. D. Penta, R. Oliveto, and D. Poshyvanyk, "Multi-objective optimization of energy consumption of guis in android apps," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 14:1–14:47, Sep. 2018. [Online]. Available: http://doi.acm.org/10.1145/3241742

[9] D. Li and W. G. J. Halfond, "Optimizing energy of http requests in android applications," in *Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile*, ser. DeMobile 2015. New York, NY, USA: ACM, 2015, pp. 25–28. [Online]. Available: http://doi.acm.org/10.1145/2804345.2804351

[10] R. Jabbarvand, A. Sadeghi, J. Garcia, S. Malek, and P. Ammann, "Ecodroid: An approach for energy-based ranking of android apps," in *Proceedings of the Fourth International Workshop on Green and Sustainable Software*, ser. GREENS '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 8–14. [Online]. Available: http://dl.acm.org/citation.cfm?id=2820158.2820161

[11] C. Sahin, M. Wan, P. Tornquist, R. McKenna, Z. Pearson, W. G. J. Halfond, and J. Clause, "How does code obfuscation impact energy usage?" *J. Softw. Evol. Process*, vol. 28, no. 7, pp. 565–588, Jul. 2016. [Online]. Available: https://doi.org/10.1002/smr.1762

[12] C. Guo, J. Zhang, J. Yan, Z. Zhang, and Y. Zhang, "Characterizing and detecting resource leaks in android applications," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov 2013, pp. 389–398.

[13] A. Nistor and L. Ravindranath, "Suncat: Helping developers understand and predict performance problems in smartphone applications," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, ser. ISSTA 2014. New York, NY, USA: ACM, 2014, pp. 282–292. [Online]. Available: http://doi.acm.org/10.1145/2610384.2610410

[14] Y. Liu, C. Xu, and S.-C. Cheung, "Characterizing and detecting performance bugs for smartphone applications," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 1013–1024. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568229

[15] M. Linares-Vásquez, C. Vendome, Q. Luo, and D. Poshyvanyk, "How developers detect and fix performance bottlenecks in android apps," in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2015, pp. 352–361.

[16] A. Sadeghi, H. Bagheri, J. Garcia, and S. Malek, "A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software," *IEEE Transactions on Software Engineering*, vol. 43, no. 6, pp. 492–530, June 2017.

[17] M. Linares-Vásquez, K. Moran, and D. Poshyvanyk, "Continuous, evolutionary and large-scale: A new perspective for automated mobile app testing," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 399–410.

[18] M. Linares-Vásquez, C. Bernal-Cardenas, K. Moran, and D. Poshyvanyk, "How do developers test android applications?" in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Sep. 2017, pp. 613–622.

[19] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann, and D. Lo, "Understanding the test automation culture of app developers," in *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, April 2015, pp. 1–10.

[20] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet? (e)," in *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ser. ASE '15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 429–440. [Online]. Available: https://doi.org/10.1109/ASE.2015.89

[21] R. L. Mace, "Ronald l. mace papers 1974-1998." [Online]. Available: https://www.lib.ncsu.edu/findingaids/mc00260/summary

[22] "Get started on android with talkback - android accessibility help." [Online]. Available: https://support.google.com/accessibility/android/answer/6283677?hl=en

[23] R. Hellman, "Universal design and mobile devices," in *Universal Acess in Human Computer Interaction. Coping with Diversity*, C. Stephanidis, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 147–156.

[24] J. Wentzel, E. Velleman, and T. van der Geest, "Developing accessibility design guidelines for wearables: Accessibility standards for multimodal wearable devices," in *Universal Access in Human-Computer Interaction. Methods, Techniques, and Best Practices*, M. Antona and C. Stephanidis, Eds. Cham: Springer International Publishing, 2016, pp. 109–119.

[25] L. Ruzic and J. A. Sanfod, *Universal Design Mobile Interface Guidelines (UDMIG) for an Aging Population*. Cham: Springer International Publishing, 2017, pp. 17–37. [Online]. Available: https://doi.org/10.1007/978-3-319-60672-9_2

[26] N. Markus, S. Malik, Z. Juhasz, and A. Arató, "Accessibility for the blind on an open-source mobile platform," in *Computers Helping People with Special Needs*, K. Miesenberger, A. Karshmer, P. Penaz, and W. Zagler, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599–606.

[27] V. Tomberg and S. Kelle, "Towards universal design criteria for design of wearables," in *Advances in Design for Inclusion*, G. Di Bucchianico and P. Kercher, Eds. Cham: Springer International Publishing, 2016, pp. 439–449.

[28] "Mobile accessibility guidelines." [Online]. Available: https://www.bbc.co.uk/guidelines/futuremedia/accessibility/mobile

[29] "Web content accessibility guidelines (wcag)." [Online]. Available: https://www.w3.org/WAI/standards-guidelines/wcag/

[30] M. Kim, E. S. Jung, S. Park, J. Nam, and J. Choe, "Application of a universal design evaluation index to mobile phones," in *Human-Computer Interaction. Interaction Platforms and Techniques*, J. A. Jacko, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 364–373.

[31] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser, "Automated accessibility testing of mobile apps," in *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*, April 2018, pp. 116–126.

[32] D. Kocieliński and J. Brzostek-Pawłowska, "Improving the accessibility of touchscreen-based mobile devices: Integrating android-based devices and braille notetakers," in *2013 Federated Conference on Computer Science and Information Systems*, Sep. 2013, pp. 655–658.

[33] Y. Zhong, A. Weber, C. Burkhardt, P. Weaver, and J. P. Bigham, "Enhancing android accessibility for users with hand tremor by reducing fine pointing and steady tapping," in *Proceedings of the 12th Web for All Conference*, ser. W4A '15. New York, NY, USA: ACM, 2015, pp. 29:1–29:10. [Online]. Available: http://doi.acm.org/10.1145/2745555.2747277

[34] Y. Mehta, A. Joshi, M. Joshi, and C. Jadhav, "Accessibility of date picker for touchscreens," in *Proceedings of the 8th Indian Conference on Human Computer Interaction*, ser. IHCI '16. New York, NY, USA: ACM, 2016, pp. 64–69. [Online]. Available: http://doi.acm.org/10.1145/3014362.3014368

[35] Z. Xie, N. Li, and L. Luo, "A study and implementation of vga multi-resolution on android platform," in *2015 International Conference on Computer and Computational Sciences (ICCCS)*, Jan 2015, pp. 110–115.

[36] L. C. Serra, L. P. Carvalho, L. P. Ferreira, J. B. S. Vaz, and A. P. Freire, "Accessibility evaluation of e-government mobile applications in brazil," *Procedia Computer Science*, vol. 67, pp. 348 – 357, 2015, proceedings of the 6th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915031257

[37] B. N. Walker, B. J. Tomlinson, and J. H. Schuett, "Universal design of mobile apps: Making weather information accessible," in *Universal Access in Human–Computer Interaction. Design and Development Approaches and Methods*, M. Antona and C. Stephanidis, Eds. Cham: Springer International Publishing, 2017, pp. 113–122.

[38] A. A. Al-Subaihin, A. S. Al-Khalifa, and H. S. Al-Khalifa, "Accessibility of mobile web apps by screen readers of touch-based mobile phones," in *Trends in Mobile Web Information Systems*, M. Matera and G. Rossi, Eds. Cham: Springer International Publishing, 2013, pp. 35–43.

[39] E. Krainz, K. Miesenberger, and J. Feiner, "Can we improve app accessibility with advanced development methods?" in *Computers Helping People with Special Needs*, K. Miesenberger and G. Kouroupetroglou, Eds. Cham: Springer International Publishing, 2018, pp. 64–70.

[40] M. C. C. Araújo, A. R. Façanha, T. G. R. Darin, J. Sánchez, R. M. C. Andrade, and W. Viana, "Mobile audio games accessibility evaluation for users who are blind," in *Universal Access in Human–Computer Interaction. Designing Novel Interactions*, M. Antona and C. Stephanidis, Eds. Cham: Springer International Publishing, 2017, pp. 242–259.

[41] J.-M. Díaz-Bossini, L. Moreno, and P. Martínez, "Towards mobile accessibility for older people: A user centered evaluation," in *Universal Access in Human-Computer Interaction. Aging and Assistive Environments*, C. Stephanidis and M. Antona, Eds. Cham: Springer International Publishing, 2014, pp. 58–68.

[42] J.-M. Díaz-Bossini and L. Moreno, "Accessibility to mobile interfaces for older people," *Procedia Computer Science*, vol. 27, pp. 57 – 66, 2014, 5th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion, DSAI 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050914000106

[43] "Get started with accessibility scanner." [Online]. Available: https://support.google.com/accessibility/android/answer/6376570

[44] "Improve your code with lint checks." [Online]. Available: https://developer.android.com/studio/write/lint?hl=en

[45] "Use node tree debugging." [Online]. Available: https://developer.android.com/guide/topics/ui/accessibility/node-tree-debugging

[46] N. Patil, D. Bhole, and P. Shete, "Enhanced ui automator viewer with improved android accessibility evaluation features," in *2016 International Conference on Automatic Control and Dynamic Optimization Techniques (ICACDOT)*, Sep. 2016, pp. 977–983.

[47] K. Moran, R. Bonett, C. Bernal-Cárdenas, B. Otten, D. Park, and D. Poshyvanyk, "On-device bug reporting for android applications," in *2017 IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, May 2017, pp. 215–216.

[48] M. Fazzini, E. N. D. A. Freitas, S. R. Choudhary, and A. Orso, "Barista: A technique for recording, encoding, and running platform independent android tests," in *2017 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, March 2017, pp. 149–160.

[49] Google, "added in api level 4 accessibilityservice. https://developer.android.com/reference/android/accessibilityservice/AccessibilityService."

[50] Android lint checks: http://tools.android.com/tips/lint-checks.