Bridging the Gap between Automated Intervention and Actual User Experience: A Mixed-Methods Study on Mobile Accessibility Issues for Screen Reader Users

Syed Fatiul Huq
University of California, Irvine
Irvine, California, USA
fsyedhuq@uci.edu

Ziyao He
School of Information and Computer Science
University of California, Irvine
Irvine, California, USA
ziyaoh5@uci.edu

Yirui He
University of California, Irvine
Irvine, California, USA
yiruih@uci.edu

Sam Malek
University of California, Irvine
Irvine, California, USA
malek@uci.edu

Abstract

Millions of people around the world experience blindness or moderate to severe visual disability, who need to rely on screen readers to perceive the content of phone screens. Guidelines and testing tools developed to aid software developers suffer from inconsistency in categorizing accessibility issues and not faithfully representing real user experience. In this paper, we aim to construct a better classification of accessibility issues, integrating feedback from screen reader users to existing computational methods. First, we conduct a systematic literature review, investigating 31 papers that demonstrated automated interventions for mobile accessibility. We juxtapose their computationally addressed issues with real user experience, by observing blind users' interaction on 4 apps across 20 user studies. Synthesizing the two studies, we construct a categorization and guideline for screen reader accessibility issues on mobile, aimed to initiate a more user-aware understanding and subsequent interventions towards accessible mobile app development.

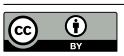
CCS Concepts

• **Human-centered computing** → **Accessibility design and evaluation methods**; **Empirical studies in accessibility**; • **Social and professional topics** → *People with disabilities.*

Keywords

Software Accessibility, Accessibility User Testing, Automated Accessibility Testing

ACM Reference Format:
Syed Fatiul Huq, Ziyao He, Yirui He, and Sam Malek. 2026. Bridging the Gap between Automated Intervention and Actual User Experience: A Mixed-Methods Study on Mobile Accessibility Issues for Screen Reader Users. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems (CHI '26), April 13–17, 2026, Barcelona, Spain.* ACM, New York, NY, USA, 27 pages. https://doi.org/10.1145/3772318.3791293



CHI '26, Barcelona, Spain

ACM ISBN 979-8-4007-2278-3/26/04
https://doi.org/10.1145/3772318.3791293

1 Introduction

According to the World Health Organization, around 2.2 billion people around the world live with visual disability [108], among whom, an estimated 43 million are blind and 295 million experience moderate to severe visual impairment (MSVI) [50]. This large population, like most people, needs to operate smart technologies to navigate the modern world. However, technologies like smartphones are fundamentally inaccessible to them, as these require visual-based perception and interaction (e.g., swipes and taps) to function. Users without visual disability can utilize visual cues to understand the interface, read text, view images and videos, locate and operate Graphical User Interface (GUI) components on the screen, and perceive progress and state changes. To make the same features accessible to the millions of individuals with full or partial blindness, alternative perception and interaction modes are a necessity.

A screen reader, an assistive technology that translates the GUI into sequential textual descriptions, offers a necessary alternative for interaction [105]. Screen Reader Users (SRU) swipe on the phone screen to navigate through GUI components, listening to the announcements of on-screen or alternative text to understand content and purpose, and double tapping on interactive components like buttons to activate their functionality. Figure 1 shows how a recipe page on mobile is visually perceived (left) and how it is perceived using screen readers (right). While screen readers transform visual content into a more accessible format, the full accessibility of mobile apps is highly dependent on whether the GUI is sufficiently programmed to expose its elements to the screen reader technology. For instance, developers need to explicitly write a content description (or alternative text) for images and icons. In Figure 1, the "favorite" functionality is provided as an icon button (top right). If "favorite" is not manually provided as a content description, the screen reader would instead announce the button as "unlabeled", making it challenging for the SRU to understand the button's purpose, missing out on the app's feature of "favorite"-ing recipes.

Such software development mistakes, caused primarily by a lack of awareness [47], contribute to the prevalent number of inaccessible mobile applications in the market [6, 26, 86, 104, 110]. For web development, Web Content Accessibility Guidelines (WCAG)

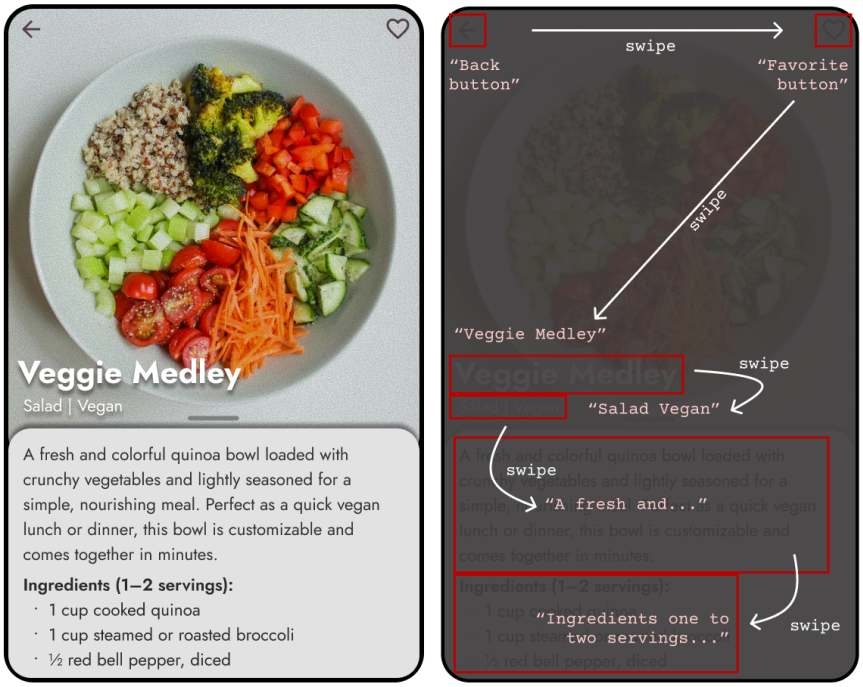


Figure 1: Comparing the perception mechanisms of mobile apps between sighted and blind users

[103] plays the foundational role of providing developers with accessibility best practices. However, their characterization of and recommended solutions to accessibility issues are specific to web technologies, which is fundamentally different from how mobile apps are implemented. While Android and Apple publish guidelines [7, 14] for their specific platforms, they are not as comprehensive and general as WCAG. In conversation with accessibility experts, Pereira et al. [91] reported that evaluators and developers seek an authoritative accessibility guideline for mobile, that can provide a comprehensive assessment of a product's accessibility and actionable steps towards resolving emerging issues. Therefore, a gap remains for a better categorization of accessibility issues for mobile, to reduce uncertainty and varied interpretations.

Furthermore, Pereira et al. [91] outlined the desire from software practitioners for automated and support tools to detect mobile accessibility issues. While automation for detecting and fixing issues has been proposed and developed, its effectiveness has been questioned by various studies. A 2018 survey of accessibility evaluation tools for mobile [94] showed that out of the 64 WCAG 2.0 success criteria, only 8 were covered by the contemporary static and dynamic tools. A 2021 systematic mapping study [69] looked at literature on evaluating software accessibility issues, either with automated tools or by expert evaluators and disabled users. They reported that automated tools only covered 20% of the total issue types in mobile. Directly comparing issues detected by blind and low vision users [23] on four Android apps with those found by two automated tools, Mateus et al. [70] showed that the tools could detect only 3 of the 39 issue types reported by actual users. On the other hand, their experiment also demonstrated that the tools could detect 11 types of issues that had not been mentioned by actual users, raising questions about the nature of these automatically detected issues. These studies echo the need to understand the discrepancies between automated interventions for detecting and resolving mobile accessibility issues and how issues actually manifest to the end user.

In this study, we attend to the aforementioned gaps in the software accessibility domain, focusing particularly on SRUs: a universal categorization for accessibility issues on mobile, and the inconsistency between automated reports and actual user experience. To do so, we aim to comprehensively map the accessibility issues addressed by existing automated techniques to their impact on users, leading to holistic understanding and specifications of individual types and categories of issues. We structure the study according to the following two Research Questions (RQ):

RQ1: What automated interventions have been studied for resolving screen reader accessibility issues on mobile?
RQ2: How do commonly analyzed accessibility issues manifest and affect screen reader users?

To answer RQ1, we conducted a systematic literature review, analyzing papers with automated interventions for mobile accessibility issues for screen reader users[1]. Our search across 5 databases, forward snowballing and manual screening yielded a total of 31 papers, from an initial set of 367. We observed four types of intervention techniques: automated crawlers, automation support, label generators and UI annotators. Across these techniques, 22 issue types

[1]For brevity's sake, we will use the term "accessibility issues" to mean mobile accessibility issues for screen reader users for the rest of the paper.

emerged under 4 categories: labeling (e.g., missing labels, inadequate description), navigation (e.g., unfocusable elements, unnatural navigation order), activation (e.g., ineffective action, complex operation) and dynamic change (e.g., latent disappearing content, latent content modification).

After establishing the categorization of accessibility issue types through the lens of automated interventions, we sought to incorporate user experiences, to better characterize those issues. For RQ2, we conducted 20 user studies with blind individuals on 4 Android apps. The testers were instructed to complete a set of tasks, which guided them through accessibility issues categorized from RQ1. We observed how these issues affected use, qualitatively analyzing their impact, perception, suggestion and workarounds. We found new, more specific issue types and a whole new category of feedback-related issues, observed how issues coalesce beyond category lines, and listed issues responsible for the most severe impacts.

In synthesizing our observations from the two studies, we were able to produce a more holistic categorization: Mobile Content Accessibility Guidelines (MCAG), for screen reader users. MCAG adopts the principles from WCAG 2.2 — perceivable, operable, understandable and robust — and incorporates common accessibility issues addressed in automated interventions and experienced by users.

Overall, the paper makes the following contributions:

- A systematic literature review of existing automated interventions for detecting and fixing screen reader accessibility issues on mobile.
- MCAG: a user-aware categorization of mobile accessibility issues for screen reader users.

The rest of the paper is organized as follows: Section 2 reviews related literature on mobile accessibility. Section 3 describes the method and results of the systematic literature review. Section 4 provides the study design and findings from the accessibility user tests. Section 5 synthesizes the findings from the studies to construct MCAG and outlines design implications and future directions, concluding in Section 7.

2 Related Work

In this section, we highlight prior work from three strands of research on software accessibility, which collectively lead to the motivation for our study.

2.1 Assessment of accessibility evaluation tools

The most relevant strand includes prior work that studied automated accessibility testing tools for mobile applications. These studies either implemented a set of existing tools [70, 93] or conducted Systematic Literature Reviews (SLRs) [69, 94] to check the quantitative coverage of those tools in testing accessibility issues.

Seibra et al. [93] implemented two Android tools, Lint [9] and Espresso [8], to show the limitations of such automation in covering accessibility requirements. They used GUAMA, a requirements guideline for developing accessible mobile applications, to conduct this experiment. Mateus et al. [70] implemented MATE [36] and Accessibility Scanner [43] on a set of 415 accessibility issue instances detected by blind and partially sighted users. The comparison for both these papers was limited to the type of issues found by either method, and did not dive deeper into the nature of those issues faced by the user. They also evaluated only two tools.

Silva et al. [94] conducted an SLR and searched the web to find academic and business tools that automatically evaluated mobile accessibility. They categorized the tools based on their technique: static, dynamic-manual exploration, dynamic-test scripts, and dynamic-automated GUI testing. Through observing documentations and samples, they also showed that only 8 out of the 64 accessibility guidelines were covered by all existing approaches. Their primary goal was to evaluate the tools' coverage compared to WCAG, with no consideration of user assessments.

Mateus et al. [69] conducted an SLR to find studies that evaluated accessibility issues in web and mobile using either of the three techniques: automation, expert evaluation and user evaluation. From the papers, automation only accounted for at most 20% of total issue types. However, they did not provide any relationship between automation and user experience beyond analyzing issue types discussed in individual literature.

These studies provide the important insight that automated tools cannot cover all the accessibility issues specified in guidelines like WCAG or observed by humans. However, since none of these studies employed actual user feedback, a gap remains in going beyond numeric comparisons of issue types, and analyzing the tools' limitations with regards to screen reader user experience.

2.2 SLRs in software accessibility

From the prior strand, we observed that accessibility testing tools can be analyzed more comprehensively with SLRs. In the second strand, we therefore look at how SLRs have been adopted to study software accessibility.

Systematic reviews and surveys have investigated accessibility across diverse software contexts. Platform-specific research has paid particular attention to mobile accessibility. These works include scoping reviews of native applications for people with disabilities [2], mappings of accessibility barriers in websites and apps [69], surveys of elderly users' needs [31], and empirical analyses of how accessibility guidelines are applied in mobile contexts [17].

Within the software development life cycle (SDLC), studies have examined how accessibility can be integrated into the broader SE process and IDEs [84], as well as how model-driven development approaches can embed accessibility requirements early in design [79]. Other reviews provide broader perspectives on accessibility practices across different phases of the SDLC [28, 40, 79, 84, 101].

Usability-oriented reviews [3, 56, 68] highlight challenges faced by specific populations, such as people with disabilities [68], visually impaired users [3], and learners in mobile learning settings [56]. These studies emphasize that current evaluation methods remain fragmented and are often limited to self-report instruments.

Finally, some reviews consider alternative data sources, such as user feedback from app reviews [78], illustrating how user-reported experiences can complement formal guidelines in practice.

These studies support SLRs as an effective method to comprehensively observe and analyze software accessibility in different domains. However, no recent SLRs covered automated techniques, for both detecting and fixing accessibility issues on mobile. Our study aims to fill this gap, and additionally incorporate end-user

perspective, which, as shown in the next strand, provide a complete picture of accessibility issues.

2.3 Understanding disability experience

For the last strand, we include papers that conducted user studies with people with disabilities to better understand software accessibility. Several papers examine accessibility in domains such as government portals [1], finances [64, 96], education [55], tourism [66], smart home IoT [33], robocall systems [92], healthcare [76], multimedia consumption [107], touchscreen devices and hybrid apps [4, 71], and public transportation [95].

Researchers have conducted extensive task-based usability tests and surveys with blind and visually impaired participants to understand pain points when using mobile interfaces. These include comparative studies contrasting experiences of sighted vs. blind users [100], assessments of video conferencing platforms [57], and evaluations of games [59, 60]. Other works adopt multi-method observational approaches, such as pilot studies of iOS applications with novice VoiceOver users [22] and comparative evaluations of multiple interface versions [106], revealing critical interface-level barriers and proposing preliminary accessibility guidelines.

Our study is aimed to complement prior work's effort to understand the disability experience by focusing on automated solutions and how their addressed issues map to blind users' experience.

3 RQ1. Systematic Literature Review

To answer our first research question, "*What automated interventions have been studied for resolving screen reader accessibility issues on mobile?*", we needed to understand to what extent prior works have experimented with and investigated such issues. For a transparent and reproducible study, we conducted a systematic literature review (SLR) following the methodology proposed by Kitchenham and Charters [54]. The methodology, adapted for SLRs in the software engineering domain, contained three phases: planning, conducting and reporting.

3.1 Planning

In the planning stage, we posed two sub-research questions to direct our SLR, and determined searching and screening strategies to find and select relevant papers.

3.1.1 Sub-Research Questions. To classify intervention methods and categorize screen reader accessibility issues for mobile, we formulated the following two sub-research questions (SRQs).

- **SRQ1:** What techniques of automated interventions have been proposed or implemented?
- **SRQ2:** What accessibility issue types have been detected or resolved by prior research?

3.1.2 Search String and Databases. We conducted a preliminary review of other SLRs in the software accessibility research domain in order to better formulate our search strategy. Based on these studies, as mentioned in Section 2.2, we combined keywords for *accessibility*, *mobile apps* and *screen readers*. To narrow down on papers with *automated interventions*, we added representative keywords. An example search string (used for the database ACM Digital Library) is as follows,

```
Abstract:(accessib* OR a11y) AND Abstract:(mobile OR android OR iOS) AND Abstract:(detect* OR test* OR automat* OR empirical* OR program* OR fix* OR repair*) AND AllField:("screen reader" OR talkback OR voiceover).
```

We also followed prior SLRs to also determine our target databases: ACM Digital Library (ACM DL) [16], IEEE Xplore [49], ScienceDirect [37], Scopus [38] and Web of Science [27]. For each database, the syntax of the search string needed to be adapted based on the database's input mechanism, but the core concepts remained consistent. However, due to imposed limits on boolean operators, keywords for *automated intervention* were omitted for ScienceDirect.

3.1.3 Screening Criteria. We established a set of exclusion and inclusion criteria to screen the papers collected from the initial search. Table 1 lists these criteria. Exclusion Criteria (EC) 01-04 concern the availability and validity of the publications. EC 05-08 and Inclusion Criteria (IC) 01-03 concern the content of the paper, determining their relevance to our proposed SRQs. We limited our search to papers published after 2009 (EC01) because that is the year the first versions of mobile screen readers were released. EC05 was determined to exclude papers that conducted empirical investigations of accessibility without providing any new technique for automated interventions. EC06 was defined to exclude papers studying other forms of disabilities or assistive technologies, for instance, switch systems for people with motor disabilities. Lastly, EC07 excluded papers focusing solely on the accessibility of web or web-based technology.

Table 1: Selection criteria for the SLR

Criteria	Description
EC01	The paper is published before 2010.
EC02	The paper is not written in English.
EC03	The paper is a duplicate or unavailable in full text.
EC04	The paper is a conference abstract or poster, or not peer-reviewed.
EC05	The paper does not provide a computational intervention for software accessibility issues.
EC06	The paper's demographic does not focus on or include screen reader users.
EC07	The paper's target platform does not include native mobile applications.
EC08	The paper is only a literature review.
IC01	The paper answers only SRQ1.
IC02	The paper answers only SRQ2.
IC03	The paper answers both SRQs.

3.2 Conducting

After the planning stage, we conducted the searches in the databases, semi-automatically detected and removed duplicates, screened the collected papers in two phases, performed forward snowballing, and analyzed the final set of papers. An overview of these steps, including the number of papers (n) selected after each search or

screening stage, is illustrated in Figure 2 following the PRISMA flow diagram for SLRs [80].

3.2.1 Search and Screening. For finding duplicates, we partially used Zotero [34], a reference management software. Zotero automatically detected potential duplicates, which we manually verified and de-duplicated.

After the initial search, the collected papers were screened in two phases, referencing the criteria listed in Table 1. In the first phase, only the title and abstract of a paper were read. One author initially read all the titles and abstracts, and a second author reviewed the excluded articles. Disagreements were listed as inclusions and moved on to the second phase. In the second phase, the full paper was read. The papers were randomly distributed between the two authors, who individually read the papers and designated exclusion or inclusion criteria. Afterwards, both authors discussed their screening results and remediated any uncertainty through further readings.

Lastly, we conducted forward snowballing, looking through articles that cited the current papers and screening based on the same exclusion and inclusion criteria.

From the initial set of 367 papers extracted from the databases, through de-duplication and screening, we ended up with 31 relevant research papers.

3.2.2 Analysis. After finalizing the 31 papers, listed as P1-P31 in Table 2, both authors read these papers and annotated them with data relevant to the SRQs.

For the first SRQ, which focused on implementation details of the automated interventions, we followed recommendations from Cruzes and Dyba [29] to conduct thematic synthesis of the papers. The process yielded a total of 30 codes in 5 categories, for instance, input modalities (e.g., 'runtime metadata', 'source code', 'screenshot' etc.), test oracles (e.g., 'WCAG', 'non-assisted interaction', etc.), and more. Details of the thematic synthesis process and the resulting thematic map is elaborated in Appendix A.1.

For the second SRQ, intended to categorize the addressed accessibility issues, we extracted all the issues mentioned in the papers and annotated them with the provided definitions, targeted UI modalities, detection heuristics and fixing methods. Furthermore, we mapped the relevant WCAG 2.2 Success Criteria [103] to each issue type, when applicable.

3.3 Reporting - SRQ1 Intervention Techniques

We observed two main purposes for the automated interventions: to test and to fix. 20 papers were for testing purposes, and 11 were for fixing. Testing interventions papers primarily consisted of tools and mechanisms to help developers detect issues in their applications. There were two types of testing interventions:

(1) *Automated crawlers*: These interventions developed a fully automated mechanism, that crawled applications without developer involvement, to find accessibility issues on the visited screens.
(2) *Automation support*: These interventions provided automated support to existing testing techniques to streamline accessibility integration.

Fixing interventions revolved around generating accessible labels or annotating accessible properties to UI elements. There were two types of fixing interventions:

(1) *Label generators*: These interventions implemented predictive models to generate textual labels for icons.
(2) *UI annotators*: These interventions annotated, automatically or with human involvement, accessibility-related properties to UI elements.

For the four total techniques, we present general patterns of their implementation details. Brief descriptions of each paper can be found in Appendix A.2.

3.3.1 Automated Crawlers. All 10 papers that developed automated crawlers implemented their tools for the Android *platform*. The target *demographic* were blind or screen reader users for 6 papers (P1, P3, P7, P15, P16, P19), while 4 also included people with low vision and motor disabilities (P2, P18, P23, P29). The latter set of papers used the Android Testing Framework (ATF) [7] as their *oracle* for issue detection. ATF provides developers with best practices to code an accessible application. These practices are based on programmatic heuristics. For instance, the `contentDescription` attribute should be provided for an image-based component, like `ImageButton`; otherwise the screen reader will announce the element as "unlabeled".

Other oracles include "non-assisted proxies" (P7, P15, P19), where screen reader accessibility issues are detected by comparing with the equivalent navigation without a screen reader. "Empirically-studied rules" (P3, P16) are used by papers where the authors generated issue evaluation rules based on their empirical study on apps and with people with disabilities. One of the most recent papers, P1, used Large Language Model as an oracle for issue detection.

The primary *analysis modality* was "runtime metadata", which, for Android, is the `AccessibilityNodeInfo`. It is an Android object encapsulating a UI component or a collection in the accessibility tree. The tree is a representation of the Android screen, based on hierarchical relationships among UI components. `AccessibilityNodeInfo` contains important data that help understand the accessibility of a component, for instance, its textual value (e.g., `text`, `contentDescription` or `hintText`), its possible functionality (e.g., `clickable`, `focusable` or `editable`), its current state (e.g., `focused`, `checked` or `selected`), its absolute and relative location (e.g., bounds, `parent` or `labeledBy`).

Different papers adopted different crawling strategies to navigate to various screens on the app under test and extract the `AccessibilityNodeInfo` of the present UI components for analysis. P23 and P29 used a semi-random crawler while P18 extracted all possible entry points from the APK to determine a more directed crawling strategy. P2 further improved this method by including non-Activity screens like pop-up dialogs, menus and more. P3, P15 and P16 designed their tools so that any crawling mechanism can be used.

3.3.2 Automation Support. Of the 10 papers on automation support, 2 were designed for the iOS *platform* and 7 on Android, while 1 was platform-agnostic. 6 tools's *demographic* were inclusive of any disability, while 3 focused specifically on screen reader users, and 1 on both screen reader and switch system users.

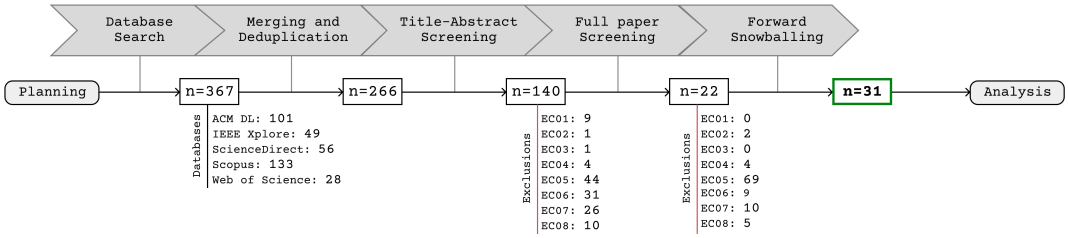


Figure 2: Overview of each phase of the systematic literature review and the associated progress of selected publications

The tools provided automation support at different *stages* of software development. Four papers targeted test executions, with P14 and P31 developing APIs for accessibility testing in code, while P12 and P21 aimed to convert existing functional unit tests or test instructions into screen reader-based execution. Two papers, P11 and P13, targeted manual testing, conducting screen reader-based execution or generating more directed reports from manual audits from developers or testers. Lastly, user testing was the domain for P4, developing an automated report generation using LLM from test transcripts with blind testers.

We observed more variety in *input modality* for the tools compared to crawlers. While some used runtime metadata (P6, P13, P15, P24), others utilized static artifacts like the source XML (P27) and source code (P14, P31), or human input like execution traces (P13), test instructions (P12) and user evaluations (P4).

3.3.3 Label Generators. While generating labels for icons can be cross-platforms, these 5 studies adopted their predictive models towards a specific *platform* to create their dataset. 4 used Android icons and 1 used iOS ones. All the *models* implemented multimodal deep learning models, constructed with a mixture of Transformer encoder-decoders and CNN classifiers. The key difference lied in their *input modality*, either depending solely on a cropped screenshot of the icon (P8, P25), or augmenting it with contextual runtime information (P22, P26) and neighbouring pixels (P17).

3.3.4 UI Annotators. Unlike label generators, these 6 papers broadened their scope in annotating other aspects and UI components. Among them, 5 worked on the Android *platform* and one on iOS. They differed in *scope*, with 2 papers (P28, P30) creating tools that support manual annotation, while the rest (P5, P9, P10, P20) fully automating the process.

For the automated approaches, we observed three predictive *models*. LLMs were used by P5 and P9, the most recent of the papers, to generate specific textual solutions. A rules-based automation, following empirical evidence, was employed by P10, while P20 employed a multimodal object detection mechanism.

In terms of *input modalities*, the common one was runtime metadata (P5, P9, P10), with P9 solely depending on screenshots. The semi-automated approaches, P28 and P30, utilized the view hierarchy at runtime along with screenshots.

3.4 Reporting - SRQ2 Accessibility Issues Analyzed

All 31 papers from our SLR analyzed one or more types of accessibility issues. However, there was an evident lack of a consistent classification of issues that all the papers referenced, echoing prior work and our motivation to construct more general categorization.

Analyzing the individual issue types across the 31 papers based on our annotations, we grouped and classified the issues into four categories and 22 types, as listed in Table 3. In this section, we summarize the issue types in the four categories: labeling, navigation, activation and dynamic change. Individual descriptions of the issues can be found in Appendix B.

3.4.1 Labeling-related Issues. We have found 9 labeling-related issues discussed in prior papers. These are issues regarding the presence and quality of alternative text to image-based or non-native UI components, which is necessary for screen readers to announce the correct information to the user. $I_{lb}1$ is the most commonly tested (11) and fixed (8) issue, related to the absence of alternative text to image-based UI. Similarly $I_{lb}2$ and $I_{lb}3$ are concerned with the proper alternative annotating for edit fields.

Issues $I_{lb}3$, $I_{lb}4$ and $I_{lb}5$ are caused by improper labeling of image-based UI and edit fields. $I_{lb}7$ and $I_{lb}8$ concern the lack information conveyed to the user about the operations of an interactive element.

These issues, primarily inferred by assessing the *contentDescription* or *hint* attributes of non-text elements and edit fields respectively, weakens the information provided to screen reader users.

3.4.2 Navigation-related Issues. We observed 6 types of navigation-related issues from our SLR. These issues address inconsistencies and failures for screen reader users to intuitively explore the UI. $I_{nv}1$ and $I_{nv}2$ are concerned with the reachability of interactive elements, caused by inappropriate attribution of `importantForAccessibility`, `accessibilityTraversalBefore` or `accessibilityTraversalAfter`, or the use of `WebViews`.

$I_{nv}3$ and $I_{nv}4$ challenge the quality of the focus order, checking whether it confuses or disorients the user. $I_{nv}5$ and $I_{nv}6$ characterize usability concerns where poor design lead to excessive interactions.

3.4.3 Activation-related Issues. There have been 3 types of issues related to activation, referring to screen reader users' action on interactive elements like buttons.

$I_{ac}1$ covers elements that are actionable to sighted users, but through screen readers, cannot be interacted with, leading to missed functionalities. $I_{ac}2$ specifically targets the implementation of `Clic`

Table 2: List of publications selected for SLR sorted by year of publication.

ID	Year	Title and Reference	Intervention Technique
P1	2025	ScreenAudit: Detecting Screen Reader Accessibility Errors in Mobile Apps Using Large Language Models [117]	Automated Crawler
P2	2025	Scenario-Driven and Context-Aware Automated Accessibility Testing for Android Apps [116]	Automated Crawler
P3	2025	Automated Accessibility Analysis of Dynamic Content Changes on Mobile Apps [72]	Automated Crawler
P4	2025	Automated Generation of Accessibility Test Reports from Recorded User Transcripts [48]	Automation Support
P5	2025	Distinguishing GUI Component States for Blind Users using Large Language Models [111]	UI Annotator
P6	2025	Towards an Inclusive Mobile Web: A Dataset and Framework for Focusability in UI Accessibility [44]	Automation Support
P7	2024	I tend to view ads almost like a pestilence: On the Accessibility Implications of Mobile Ads for Blind Users [45]	Automated Crawler
P8	2024	Dark-Side Avoidance of Mobile Applications With Data Biases Elimination in Socio-Cyber World[65]	Label Generator
P9	2024	Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM [63]	UI Annotator
P10	2024	Don't Confuse! Redrawing GUI Navigation Flow in Mobile Apps for Visually Impaired Users [112]	UI Annotator
P11	2024	Towards Automated Accessibility Report Generation for Mobile Apps [98]	Automation Support
P12	2024	AXNav: Replaying Accessibility Tests from Natural Language [99]	Automation Support
P13	2023	Assistive-Technology Aided Manual Accessibility Testing in Mobile Apps, Powered by Record-and-Replay [89]	Automation Support
P14	2023	Early accessibility testing – an automated kit for Android developers [41]	Automation Support
P15	2022	Groundhog: An Automated Accessibility Crawler for Mobile Apps [90]	Automated Crawler
P16	2022	Too Much Accessibility is Harmful! Automated Detection and Analysis of Overly Accessible Elements in Mobile Apps[73]	Automated Crawler
P17	2022	Towards Complete Icon Labeling in Mobile Applications [25]	Label Generator
P18	2022	Accessible or Not? An Empirical Investigation of Android App Accessibility [26]	Automated Crawler
P19	2022	Automated Detection of TalkBack Interactive Accessibility Failures in Android Applications [5]	Automated Crawler
P20	2021	Screen Recognition: Creating Accessibility Metadata for Mobile Applications from Pixels [113]	UI Annotator
P21	2021	Latte: Use-Case and Assistive-Service Driven Automated Accessibility Testing Framework for Android [88]	Automation Support
P22	2021	Data-driven accessibility repair revisited: on the effectiveness of generating labels for icons in Android apps [74]	Label Generator
P23	2020	Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward [6]	Automated Crawler
P24	2020	An Epidemiology-inspired Large-scale Analysis of Android App Accessibility [86]	Automation Support
P25	2020	Unblind your apps: predicting natural-language labels for mobile GUI components by deep learning [24]	Label Generator
P26	2020	Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements [61]	Label Generator
P27	2019	Development of Automatic Evaluation Tool for Mobile Accessibility for Android Application [81]	Automation Support
P28	2018	Robust Annotation of Mobile Application Interfaces in Methods for Accessibility Repair and Enhancement [115]	UI Annotator
P29	2018	Automated Accessibility Testing of Mobile Apps[36]	Automated Crawler
P30	2017	Interaction Proxies for Runtime Repair and Enhancement of Mobile Application Accessibility [114]	UI Annotator
P31	2017	A New API for Android Accessibility Testing [30]	Automation Support

kableSpans, which can cause $I_{ac}1$. Lastly, P16 describes $I_{ac}3$ which unintentionally exposes screen readers to hidden features.

These issues, stemming from inappropriate rendering of interactive elements, block screen readers accessing app functionalities.

3.4.4 Dynamic Change-related Issues. Lastly, we categorize 5 types of dynamic change-related issues, following P3, where UI elements are moved or modified without notifying the screen reader users.

$I_{dy}1$, $I_{dy}2$ and $I_{dy}3$ are issues caused by dynamically appearing and disappearing contents in the screen, without notifying

Table 3: Accessibility issue types observed in the SLR, categorized into four categories, presented with a short description, related WCAG 2.2 Success Criteria [103], and list of papers that tested and fixed the issue. Ellipses ("...") indicate no direct mapping to a Success Criteria or the respective category.

ID	Issue Type	Description	WCAG	Tested In	Fixed In
Labeling-related issues					
$I_{lb}1$	Missing label	No alternative text for non-textual widgets.	1.1.1, 4.1.2	P1, P2, P7, P13, P14, P18, P23, P24, P27, P29, P31	P8, P17, P22, P25, P26, P28 P30
$I_{lb}2$	Missing hint	No hint for input fields.	3.3.2	P2, P13, P14, P18, P23, P24, P27, P29	P9
$I_{lb}3$	Editable content description	Input fields have their label in content description.	3.3.2	P2, P18, P23, P24, P29	…
$I_{lb}4$	Redundant description	Repeating the type or action hint of element in the label.	…	P1, P2, P18, P23, P24	…
$I_{lb}5$	Inadequate description	The label of an element provides incorrect or incomplete information	2.4.6	P1, P13	P28, P30
$I_{lb}6$	Duplicate labels	If more than one non-textual clickable widget on the same screen have the same label.	…	P1, P2, P18, P23, P29	…
$I_{lb}7$	Unsupported class name	A custom View that does not provide the type of elements to screen reader.	4.1.2	P2, P18, P23, P24	P28
$I_{lb}8$	Missing interaction type	When assistive services have no programmatic access to whether and what action can be performed on UI elements.	4.1.2	P13, P14	…
Navigation-related issues					
$I_{nv}1$	Unfocusable interactive element	When a screen reader cannot focus on an interactable element via swipe gestures.	2.1.1	P6, P7, P12, P13, P15, P19, P21	…
$I_{nv}2$	Navigation loop	Cyclical navigation on a subset of UI elements.	2.1.1	P12, P21, P23	…
$I_{nv}3$	Unnatural navigation order	Where directional navigation does not follow a logical order.	2.4.3	P15	P20, P28, P30
$I_{nv}4$	Unapparent focus switching	Sudden changes in navigation sequence direction.	2.4.3	…	P10
$I_{nv}5$	Excessive interactions	Design that leads to excessive swipes to reach an element with directional navigation.	…	P7, P15, P16, P21	P20
$I_{nv}6$	Broken up text	When each word of a paragraph is focused individually.	…	P13, P21	…
Activation-related issues					
$I_{ac}1$	Ineffective action	When screen reader is unable to perform an action with default double tapping.	2.1.1	P7, P13, P15	…
$I_{ac}2$	Clickable span	Text spans configured as clickable.	2.1.1, 4.1.2	P23, P29	…
$I_{ac}3$	Overly actionable	Elements that expose action to screen reader, otherwise invisible on screen.	…	P16	…
Dynamic Change-related issues					
$I_{dy}1$	Latent appearing content	An element that initially is not present on a loaded window, but appears a few moments later and remains on the screen.	…	P3	…
$I_{dy}2$	Latent disappearing content	An element that disappears either after a set period or as a result of user interaction.	…	P3, P15, P21	…
$I_{dy}3$	Latent short-lived content	An element that initially is not present on the screen but appears and remains on the screen only for a brief duration.	…	P3	…
$I_{dy}4$	Latent moving content	An element that is initially visible on the screen but subsequently gets relocated to a different part of the screen.	…	P3	…
$I_{dy}5$	Latent content modification	An element that remains on the screen but its attributes change.	4.1.2	P3	P5, P20

the screen reader. $I_{dy}4$ is similar, but for elements moving from one bounds to another. Lastly, $I_{dy}5$ is concerned with components changing states due to user action, but conveying the change only visually.

These issues, resulting from inadequate annotation of the `accessibilityLiveRegion` attribute, limit screen reader users' awareness of dynamically relocating or modifying elements on the screen.

3.5 Discussion

3.5.1 Intervention Trends. From analyzing the intervention techniques and accessibility issues they detected, we were able to observe certain trends in automated interventions.

Firstly, for both testing and fixing interventions, we observed the shifting trend of **input modalities**, from static source code to runtime metadata and screenshots. This shift in analysis mode, from **static to dynamic**, provides not only a better and more high fidelity snapshot of an app's behavior, but also diminishes the dependence on proprietary source code for accessibility testing. Real-world adoption of these intervention techniques becomes easier and more scalable, needing only the access to the public executable files, e.g., APKs of Android apps.

For testing techniques, there were two main detection mechanisms: programmatic rules and assisted proxies. Multiple tools referenced ATF, Android's accessibility guidelines, as the **programmatic rules** for inaccessible elements. Tools that adopted this as their oracle included automated crawlers (P2, P18, P23, P29), testing frameworks (P31), and semi-automated evaluators (P24). While convenient for integrating into crawling processes and conducting large-scale empirical studies, these tools were usually limited to labeling-related issues.

To counter this limitation, we saw the advent of **assisted proxies**, where tools compared navigation and interaction differences between touch-based and screen reader-based exploration. These mechanisms were able to detect previously untestable issues related to navigation and activation. Both P15 and P13 emulated, at runtime, screen reader navigation, albeit through different means: automated crawling and converting manual testing respectively. Both of them, and P7, which deployed P15 for advertisements, were able to detect unfocusable elements and ineffective actions. On the other hand, P19 generated UI flow graphs from navigating the app to detect unfocusable elements. P10 also used UI flow graphs, then redrew problematic and subpar navigation flows to resolve unnatural navigation order.

Lastly, for fixing solutions, we observed two main use cases: human-level and system-level. P28 and P30 placed the purpose of their semi-automated annotation pipelines at **human-level**. They proposed a social resolution to accessibility issues, with their solutions to be used to fix apps through crowdsourcing. Human annotators would annotate problematic UIs and publish them for screen reader users.

Recent solutions delved into automating this process, either through pixel-based (P8, P17, P20, P25) or context-aware (P5, P9, P10, P22, P26) predictive models. The target use case for their solutions was **system-level**, as the generated annotations can be used by the operating system to annotate inaccessible apps at runtime.

Takeaway 1

Automated interventions for mobile accessibility are shifting away from programmatic heuristics-based and crowd-reliant techniques to high-fidelity automations for more accurate and scalable solutions.

3.5.2 Subjectivity in Issues. In analyzing the issue types and how they have been detected, we observed that a subset of them contain subjectivity in their characterization. These pose challenges to existing automated tools in correctly detecting and resolving them.

Automation has been effective in detecting issues with **definitive rules**. For instance, missing label ($I_{lb}1$) and missing hint ($I_{lb}2$) issues are only dependent on whether there are no alternative text or hints. Unfocusable ($I_{nv}1$) or inactionable ($I_{ac}1$) elements can be definitively determined by comparing different navigation methods.

For non-definitive ones, automation techniques usually employ **approximate rules**. For instance, to detect duplicate labels, P24 checks whether there are at least two `clickable` elements in the same screens. However, that did not determine whether the elements are duplicate by design. Excessive interactions ($I_{nv}5$) is dependent on the rule of "swipe number over 15", but this does not definitively determine whether it is perceived as an issue to the user during actual use.

Another technique employed has been to rely on **human inference**. P13 depends on developers to go through the generated report and infer whether the screen reader output matches their intended presentation to the user. This was used to address, for instance, inadequate description ($I_{lb}5$) and unnatural navigation order ($I_{nv}3$). The mechanisms from P28 and P30 relies on human understanding to provide better annotations.

Some fixing strategies (P6, P20) base their solution on **empirical rules** to resolve focus-related issues ($I_{nv}3$). Their techniques group and segment UI elements to better suit heuristics derived from human feedback.

Newer techniques are looking towards **LLMs** for solving this limitation, as LLMs have shown capability in understanding subjective issues. P1 showed that LLM can detect inadequate descriptions (I_{lb}) while P5 and P9 generated hints for UI states ($I_{dy}5$) and edit fields ($I_{lb}2$), respectively, using LLM. LLM-based solutions to detect subjective accessibility issues have been implemented on the web [46], and therefore poses as a promising future direction in the mobile accessibility domain.

Takeaway 2

A major limitation of existing automated techniques is their inability to detect and fix subjective accessibility issues.

4 RQ2. User Study

In answering RQ1 in the previous section, we were able to categorize and specify types of accessibility issues addressed by prior techniques. However, this categorization was limited by the technical definitions and processes employed by those techniques, which are prone to missing or mischaracterizing actual issues experienced by screen reader users [23, 69, 70, 94]. To complete the effectiveness of our categorization, and augment the limitation of automated interventions, we needed to understand the experiential specifications of these issues.

In this section, we aimed to answer RQ2, "*How do commonly analyzed accessibility issues manifest to and affect Screen Reader users?*" To do so, we conducted user studies with blind participants on inaccessible mobile applications. The study with human subjects had been approved by the Institutional Review Board (IRB) of the authors' organization.

4.1 Study Design

4.1.1 Applications Under Study. To decide on the platform and apps to test, we took insights from RQ1. First, since we observed that the predominant platform focused on in prior work was Android (27/31), we chose Android projects as our test artifacts.

Next, with the aim to investigate the issue types characterized and categorized in RQ1, we utilized the open-source nature of Android to inject accessibility issues into apps. However, in our pilot studies with a finance app named Money Manager (A_{mm}) [75], we found that incorporating too many types of issues can get overwhelming for the tester to cover. Furthermore, injected issues may not reliably emulate how inaccessibility is manifested in the real world. Therefore, we decided to instead select apps based on the following selection criteria.

(1) The app contains issues from the two popular trends, as specified in Section 3.5.1: programmatic rules (ATF for Android) and assisted proxies.
(2) The app contains at least one issue type from the first three categories, as listed in Table 3 (We excluded the category of dynamic change-related issues as neither of the trends in the first criterion focused on them).
(3) The category of the app is popular and commonly used [19].

We searched and manually evaluated apps from prior work [45, 48] and online curated lists [77, 82]. Our final selection included four Android apps: Money Manager (A_{mm}) [75], uHabits (A_{uh}) [52], AnkiDroid (A_{ak}) [12] and Aard2 (A_{ar}) [53], listed with relevant information in Table 4. The apps included both large and popular projects, along with smaller ones. Pulled from different categories, they contained common usage scenarios, like submitting forms, viewing listed entries, reading content, personalization and more.

The full set of known issues in the selected apps, existing or injected, is listed in Table 12 of Appendix B.1. Other than A_{mm}, which contained injected issues, we relied solely on the existing issues in the apps. There was one exception with A_{uh}, where we injected one Ineffective Action ($I_{ac}1$) issue type to satisfy our selection criteria of including at least one issue from each category. Table 6 contains a subset of issues for each app as examples.

4.1.2 Participants. We recruited our participants from two sources. Primarily, we collaborated with Fable Tech Labs Inc. [39], an accessibility testing organization that recruits disabled testers to evaluate websites and mobile apps. Second, we recruited from Program-L [83], a forum for visually impaired programmers. All participants were compensated for, either through the employing platform (Fable) or with $50 gift cards (Program-L) for each session.

Table 5 provides information of the testers. All participants were blind, residing in North America (US and Canada), with experience in using TalkBack on Android. No participants tested the same app twice, and reported to not have used the app before. While some participants tested more than one app, the test sessions for each app were conducted separately, preventing any cross-app influence.

4.1.3 Study Protocol. Test sessions were conducted individually for individual applications and lasted for at most one hour. The protocol began with an introduction and reception of their informed consent, followed by a brief description of the app under test. Before providing the tasks, participants were given a scenario, imagining they had installed the app on their own accord and would now explore the app without explicit instructions. This provided us with the user's natural navigation style, without being biased by our task specification. They were asked to follow the Think Aloud protocol [102], where they would speak out loud what they were doing and why, and if faced with challenges, how they were perceiving the issue and how they would work around it.

Once they finished with their navigation, we instructed them to complete the remaining tasks from our predetermined task list. For each app, we designed tasks so that the participants needed to navigate through and/or interact with inaccessible UI components. Examples of tasks associated with an issue are listed in Table 6. For instance, in A_{ak}, the task to "Add a new (flash)Card" would require the tester to find and activate the "Add" button — which is unlabeled — and in the form to create a new card, they would encounter the unlabeled Expand/Collapse buttons. The detailed protocols with scenarios and task lists are provided in Appendix B.2.

When the user encountered an accessibility issue, we first let them overcome the issue by themselves. This provided us with a more candid observation of the impact of an issue. We provided hints as a last resort for them to complete the tasks. Throughout the process, we asked clarifying questions about their experience with the accessibility issues.

We conducted five user tests on each of the apps, following 1-2 pilot sessions. We used the pilot sessions to review and refine our protocol, checking whether the testers were navigating to the intended screens and UI elements.

The studies were conducted remotely over Zoom. Participants joined from both their Desktop computer and Android device. The screen was shared from their phone while the computer picked up audio of both the tester and the phone's TalkBack announcements. The recording and transcript were collected for analysis.

4.1.4 Analysis. We analyzed the recordings using Thematic Analysis, following Braun and Clarke's approach [18]. Two authors were involved in the coding process, *familiarizing with the data* with repeated viewing of the recordings across subsequent coding steps. To code the recordings, we first extracted blocks of recordings that talked about an individual issue. The same issue might be mentioned or encountered multiple times from repetition of tasks or retrospective review by the tester. All blocks were put under the common issue instance.

On these blocks, we conducted a hybrid coding mechanism employing both deductive and inductive methods. The deductive coding was aimed at classifying the issue, based on the issue types we categorized from the SLR, as listed in Table 3. We conducted inductive coding on the testers' behavior and speech, helping us understand how they were perceiving and being impacted by the issues. In conclusion, our *initial code* consisted of the existing issue categorization and new codes, both of which the coders compared among and removed discrepancies through discussion.

From the codes, *themes were observed and denoted*, grouping similar codes together. Furthermore, we *reviewed the themes* by analyzing codes within and across the four apps, the different interviews and different issue instances. We *finalized the themes* by defining and exemplifying them, resulting into four themes: "Impact", "Perception", "Suggestion" and "Workaround". We list these

Table 4: Details of open-source Android applications included in the user study.

App	ID	Domain	Core Tasks Evaluated	Stars	Latest Update
Money Manager	A_{mm}	Finance	Creating and tracking expenses in different categories.	6	Feb 2021
uHabits	A_{uh}	Health and Lifestyle	Creating and updating different types of habits.	9k	Aug 2025
AnkiDroid	A_{ak}	Education	Creating, importing and practicing flash cards.	10k	Sep 2025
Aard2	A_{ar}	Encyclopedia	Importing, reading and bookmarking articles.	511	Jul 2024

Table 5: Tester information from the user studies (anonymized).

ID	Gender	Platform	Apps tested
T1	Male	Fable	A_{mm}
T2	Male	Fable	A_{mm}, A_{ar}
T3	Female	Fable	A_{mm}, A_{uh}
T4	Female	Fable	A_{mm}
T5	Male	Fable	A_{mm}
T6	Female	Fable	A_{uh}, A_{ak}, A_{ar}
T7	Female	Fable	A_{uh}, A_{ak}
T8	Female	Fable	A_{uh}
T9	Male	Program-l	A_{uh}
T10	Male	Fable	A_{ak}
T11	Male	Program-l	A_{ak}, A_{ar}
T12	Male	Program-l	A_{ak}, A_{ar}
T13	Male	Fable	A_{ar}

themes, along with the deductive coding category of "Issue", in Table 7, with description, sample codes and example scenarios. Detailed codebook for the four themes is provided in Appendix B.3.

4.2 Findings

From the accessibility tests, we found various categories and types of issues observed by and affecting the testers in a variety of manners. In Table 8, we present the numeric prevalence of six issue categories, including a newly categorized one (detailed in Section 4.2.6). Here, each count represents an instance of an individual being impacted by an issue. We see that mislabeling (81) and navigation-related (89) issues appeared most often. For each category, we also list the proportion of each impact. We see that the most common impact is *confusion*, which often led to *incorrect use* and *inefficiency*. Activation-related issues were the most severe, with 43.33% causing *blockers*, followed by dynamic change and feedback-related issues primarily causing *obstructions*.

To present the qualitative findings, we segment this section based on the issue categories from Table 3 and demonstrate how these issues manifested to and impacted the users.

4.2.1 Labeling-related issues - missing labels. In A_{mm}, FAB buttons for adding Budget Category and Budget Plan were unlabeled ($I_{lb}1$). For the latter, most users were *unbothered*, as they had already experienced a similar layout when adding a budget category. T4 explained, "I went in there before, and that's how I know (that this is the add button), otherwise I never would have known that."

For the former, OCR helped two users guess the functionality, as it announced "Detected icon plus". However, most[2] were *bothered* by the lack of a label and *suggested manual labeling* by developers. For one, it *caused inefficiency* as they explored the page multiple times to make sure there was no other options to add a category.

In A_{uh}, there were decorative icons during the onboarding process, which were either announced with generic labels like "graphic image" ($I_{lb}5$) or had no announcement ($I_{lb}1$, $I_{lb}8$). T2 said regarding the latter, "Talkback indicated auditorily that there's something there, but it didn't announce the content or the type of element." Most testers were able to deduce with *contextual clues*, after a phase of *confusion*, that these were merely decorative.

There was a color picker button when adding a new Habit, that was unlabeled ($I_{lb}1$). Three testers guessed its functionality based on the neighbouring text label. One completely ignored it while another (T8) *incorrectly understood the functionality*, "I assume it must be for somebody who has visual impairment that's color-based. It may change the color so they can see the screen better."

We observed how missing labels can cause *obstructions*, when icons to mark daily progress of a Habit were unlabeled. All testers *needed hints* to understand their functionality and activate them. OCR was not able to accurately describe the icons, incorrectly announcing "close" for example. T6 feared of *unintended outcomes*, "I wouldn't quite feel comfortable interacting with them, just because I wouldn't want to mess anything up."

In A_{ak}, testers faced an unlabeled FAB icon intended to add a new Card. While *bothered* and *confused*, all of them were able to *guess its functionality* with the combination of OCR announcement ("Detected icon plus," and a preceding text that said, "Start adding cards using the '+' icon."

Some testers faced *obstruction* with an unlabeled collapse button when editing a new Card. OCR only announced "up arrow" and users were unable to guess that the button would collapse the associated edit field.

The worst case of *obstruction* from missing labels was observed in A_{ar}, where the five tab headers to navigate to primary features of the app were unlabeled. All users constantly needed hints to guess

[2]We refer to the prevalence of an issue instance with "all", "most" and "some" when all, more than two or less than three of the five testers for that app reported the mentioned impact of the issue, respectively.

Table 6: Sample issues injected (denoted with *) or existing in apps, with associated UI components, screens, and the task instruction provided to reach each issue.

App	Issue Type	Issue Description	UI Component(s) - Screen	Task Instruction
A_{mm}	Duplicate labels ($I_{lb}6$)	*Two edit fields have the same ambiguous hints, "Enter Info."	Product Name and Price EditText hints - Add New Transaction page	Add a new expense.
	Inadequate description ($I_{lb}5$)	*Button to add a "Category" is labeled "Add Income."	Add Income Button - Budget Category Info dialog	Add a new expense category.
	Clickable span ($I_{ac}2$)	*"Visit our website" contains a clickable span on the word "website."	"Visit our *website*" text - Budget Analysis page	Check your budget analysis → Explore for more information."
A_{uh}	Unnatural navigation order ($I_{nv}3$)	Labels for edit fields to create a habit, are focused after the field instead of before.	Create Habit page	Create a new habit.
	Ineffective action ($I_{ac}1$)	*Customized delete button requires extra long press to complete activation.	Delete button - Edit Habit page	Check details of your habit → Delete this habit.
A_{ak}	Missing label ($I_{lb}1$)	No label for floating action button to add a new card.	Add button - Home/Decks Page	Import cards or add a new card.
		Buttons to expand or collapse edit fields are not labeled.	Expand/Collapse button - New Card Edit page	Add a new card.
	Unfocusable Interactive Element ($I_{nv}1$)	Flash cards are displayed as a webview element that cannot be focused on consistently.	Flash Card view - Card practice page	Practice the flash cards in a deck.
A_{ar}	Missing label ($I_{lb}1$)	The headers of each tab in the app is unlabeled	Tab headers - Home page	Open and explore the app.

Table 7: Summary of themes identified through thematic analysis, with descriptions, example codes and scenarios.

Theme	Description	Example Codes	Example Scenario
Issue	Characterizing the issue faced by testers based on their description and potential computational cause.	"Missing label", "Unfocusable element", "Unnatural navigation order".	
Impact	To what extent the issue affected the user's completion of task or navigation of the app.	"Blocker", "Obstruction - required hint", "Caused inefficiency".	"I think I would tap here, but because it's not identified as a button or a link, I don't exactly know."-T6, leading to obstruction or confusion.
Perception	How the tester perceived a problematic UI component or feature, especially when their perception is different from the computational reason.	"Incorrect understanding of functionality", "Incorrect understanding of information".	"I'm not sure what it's looking for."-T2, the tester thought they needed to add a category for an edit field's hint to change.
Suggestion	Instances where the tester provided recommendations for better design and implementation to alleviate an issue.	"Instruction desired", "Alternative interaction mode".	"What I would expect to see is have it tell me what I'm supposed to enter in there."-T3, on ambiguous hints on an edit field
Workaround	When the tester executes or mentions a possible workaround to circumnavigate an issue.	"Restart the app", "Customized gesture".	T3 said they usually double tap and hold for a second when a button does not work just double tapping

Table 8: Proportional prevalence of issue categories across different impact levels. The count is tallied from all four apps for each instance of an issue within that category, observed by an individual tester. Impact with the highest prevalence for each category is presented in bold.

Impact	Missing Label	Mislabeling	Navigation	Activation	Dynamic Change	Feedback
Blocker	0.00%	0.00%	10.11%	**43.33%**	0.00%	0.00%
Obstruction	17.14%	14.81%	13.48%	16.67%	**100.00%**	**29.73%**
Unusable	0.00%	0.00%	0.00%	16.67%	0.00%	0.00%
Incorrect use	0.00%	6.17%	5.62%	0.00%	0.00%	0.00%
Caused inefficiency	2.86%	2.47%	22.47%	3.33%	0.00%	18.92%
Confusion	**31.43%**	**41.98%**	**23.60%**	20.00%	0.00%	24.32%
Bothered	22.86%	7.41%	6.74%	0.00%	0.00%	10.81%
Unbothered	25.71%	27.16%	17.98%	0.00%	0.00%	16.22%
Total #	35	81	89	30	11	37

which unlabeled button represented which page, perceiving them as *blockers* from further using the app.

> While some missing labels were understood through nearby context clues and auto-generated labels, blind users were predominantly confused and obstructed due to unlabeled buttons.

4.2.2 Labeling-related issues - mislabels. In A_{mm}, when adding a new Expense, users were faced with two edit fields with the same ambiguous hints, "Enter Info." With their corresponding textual labels, "Name" and "Price", unfocusable, some users were *confused*, *incorrectly understanding their purpose*. Two users *incorrectly used* the edit fields to write a Category, which is also induced by the preceding dropdown menu.

The dropdown menu with the label "Category" was intended to select a Category for the current Expense. However, in the beginning, it was empty, as users were expected to add them manually. Without that expectation conveyed to the user, the empty dropdown list *caused inefficiency* (1/5), *confusion* (1/5) and *obstructions* (2/5). T4 vocalized their confusion, "So I'm double-tapping on the drop-down list. And nothing's happening... I wonder do I just write it in then?" This led them to incorrectly fill up the edit field.

A_{uh}contained at least 8 instances of mislabeling issues, spanning insufficient hints and uninformative labels ($I_{lb}5$), duplicate labels ($I_{lb}6$) and missing interaction type ($I_{lb}8$). To add a Habit, users first needed to choose the type of Habit — "Yes or No" or "Measurable". These were presented as custom views with no specification of the interaction type, causing *confusion* and *incorrect understanding* for most. Then, in the Habit adding form, users faced with insufficient hints for some fields, which was worsened by an unintuitive navigation flow, causing *confusion* (5/5) and *incorrect use* (2/5) of the edit fields.

After adding their Habits, users were presented with a list of added Habits, which were not marked as interactive. So users did not know that they were meant for the open Habit Details page. In that page, there were two "Edit" buttons, one to edit the Habit, and one to edit a calendar. T2 said, "There's this standard edit button. But then there's another edit button. So what is it? What am I editing when I tap that?"

A_{ak}also consisted of uninformative labels and missing interaction types. In the home screen, a button intended to sync Cards with the user's online profile was unintuitively labeled as "Sync Log In". T10 *incorrectly understood* its functionality, "I'm guessing it's to log in, but the labeling isn't the most clear." Similarly, when adding a Card, users found a "make field back sticky" button. They perceived this button as *meaningless*, causing an obstruction.

When practicing the Cards, there were multiple mislabeling inaccessibility. First, when focusing on the Card, Talkback announced some random numbers, meant to convey daily progress. These were *confusing* and *meaningless* to some users. To check the answer for a card, users needed to click on the "Show Answer" button. While it did not identify the type, the labeling was clear enough to most users to be deemed as an interactive element. After the answer is shown, users need to choose the difficulty level, "easy", "hard" and so on. Not only were these buttons unidentified as buttons, they had corresponding labels "3d", "1m" and more to signify the days ("d") or minutes ("m") needed as intervals before the next practice. These abbreviated labels were *meaningless* to the user, leading to *obstructions*. Two users *needed hints* to proceed.

> Lack of proper labeling caused blind users to incorrectly understand and use edit fields and buttons, sometimes leading to obstruction from an additional lack of type specification.

4.2.3 Navigation-related issues. Users experienced multiple types of navigation-related issues in A_{mm}, from excessive interactions ($I_{nv}5$) and unnatural navigation orders ($I_{nv}3$) to unfocusable elements ($I_{nv}1$). The excessive interactions manifested in the Add Expense page. To submit a new Expense, users needed to traverse through all the dates in a calendar view to reach the submission button below. Not only did this *cause inefficiency* (4/5), but for some an *obstruction*, since they were unable to find the button without hints. Suggestions to fix this issue included *changing the layout* to put the calendar as its own page, and *relocating the button* to the top for easier reachability.

Users experienced unfocusable elements when navigating charts. Rendered in `Webviews`, the charts either announced *meaningless* numbers or did not expose any of the underlying elements to the screen readers, causing *blockers* (5/5). For a more accessible chart, T1 suggested, "A good alternative would be a table."

A significant navigation-related issue in A_{uh} occurred when filling out a new Habit form. The edit boxes were designed to announce the corresponding text labels after the box itself, leading to *confusion* and *incorrect understanding* for all testers, regarding what information to enter where. When viewing the Habit list, the headers denoting dates were not focusable, adding to the users' *confusion* about the meaning behind the unlabeled progress icons. Similar to A_{mm}, the Habit details contained inaccessible charts.

Adding a new Card in A_{ak} presented the users with an unintuitive layout, which caused two issues. Firstly, the focus went to an edit field directly, instead of the top of the page. Because of this, three users missed the "Save" button placed at the top, later leading to *inefficiency* and even *obstruction* to save the Card. Furthermore, the distance between an edit field and its associated text label was populated by multiple buttons, some suffering from labeling issues. This caused one tester *confusion* in understanding which edit field they were focusing on.

During Card practice, the `WebView` representation of the Cards caused multiple navigation-related issues. For instance, it hindered focus from going to text fields that showed the answer or the next question. Almost all users *incorrectly understood the layout* of the page and needed *multiple iterations* to grasp it. T6 said, "Seems to be kind of erratic. Sometimes it lands on the question, sometimes it lands on the drawer button. It would be really super nice if [...] Talkback automatically landed on it (the answer or question) and then automatically just read it out loud."

In A_{ar}, we observed an instance of over-accessibility. When opening a Wiki article page, users navigating the page with swipe navigation found themselves in an incorrect page. This was caused by the page loading up adjacent articles in the background, invisible to sighted users. This phenomenon *caused inefficiency*, having to swipe through a whole irrelevant article to reach to selected one. T12 *misunderstood the state*, saying, "I thought the page was like refreshing between the articles." Users either needed sighted hints to continue (3/5), or deduced themselves using *explore by touch* (1/5) or *double finger swipes* (1/5).

> Unnatural focus orders and unreachable information confused blind users about the layout and functionality of the UI, while unreachable interactive elements blocked them from using the app's features.

4.2.4 Activation-related issues. In A_{mm}, one user experienced ineffective action ($I_{ac}1$) when trying to write in a non-native edit field, which was not activating their keyboard. One faced an over-actionability issue ($I_{ac}3$) in the form of a slider, intended for a visual demonstration of budget usage to sighted users. T3 feared that the slider could cause *unintended outcome*, "It's easy to mess with that slider." When encountering a clickable span ($I_{ac}2$), some users were able to visit the link, while others either could not focus on it or it would not activate the link, leading to *blockers*.

The "Settings" button in A_{mm} was inactionable. While an obvious *blocker*, it also created *confusion* and *inefficiency* as none of the users were able to determine whether the button worked or not.

To set a reminder for a new Habit in A_{uh}, users were faced with an inaccessible time picker. The custom view lets sighted users select the hour and minute by sliding a virtual clock's arrows. For screen readers, it was *unusable*. T8 said, "I tried swiping up and swiping down, because it's a slider. But it doesn't do anything." Users preferred standard time picking mechanism through dropdowns or text entries.

> Custom interactable elements are not actionable to blind users, preventing them from using app features.

4.2.5 Dynamic change-related issues. To delete a Habit in A_{uh}, the customized Delete button required a long press. However, the progress for the press is only shown visually, through color changing. Without auditory or haptic feedback, the tester perceived the button as a *blocker*.

A similar issue was present in A_{ak}, with the button for adding Cards required a long press to open a submenu. Not only was the long press progress unannounced, once activated *through hints*, no text announcement was provided for the new submenu. T7 said, "I don't know if it's doing anything because I didn't get any feedback ... it was just dead silence. I don't know if it added anything or not." This resulted in *obstructions*, where none of the testers were able progress without repeated hints.

In A_{ar}, the bookmarking button for a Wiki article only changes color after activation, without providing any textual modification. It caused *obstructions*, as no user could confirm whether they were able to correctly bookmark an article.

> Dynamic changes that only rely on visual modifications are perceived as failed functionality to blind users.

4.2.6 Feedback-related issues. This new category of issues emerged from the user study, regarding the lack of proper messaging from the app preceding required tasks or following completed ones.

In A_{mm}, a common issue rose in the app's failure to convey to the user that they needed to first add Categories and an Income to be able to add a new Expense. This resulted in severe *inefficiencies* as almost all the users had to redo the process every time their add feature would fail. It also *confused* them about the nature of the empty dropdown button. The lack of guided and properly located instructions led to *obstructions*, where three users failed to complete the task. "So I did all this work, and it still did not add (the transaction). I don't know what I'm doing wrong here." (T3)

After selecting a color from the color picker in A_{uh}, there was no textual feedback indicating that a color was added, leading to *confusion* whether the action was successful.

In A_{ak}, no textual feedback was announced after users pressed the "save" button. They had to traverse the page again, find that the edit fields have been reset, to realize what had happened. It caused *confusion* (3/5) and *inefficiency* (1/5). T12 suggested, "I really would like to have heard 'card saved' or 'card one saved' or something."

Similarly, in A_{ar}, importing a dictionary to the app was not followed by any confirmation. One tester *incorrectly understood* that no dictionary was added, and had to *inefficiently* go to and traverse the "Dictionaries" tab to confirm manually.

> Lack of feedback in textual or auditory form after selection or submission functionalities leaves blind users confused about the outcome of their action.

4.3 Discussion

4.3.1 New issues. In Table 9, we list the new issue types we found from our user tests. This includes not only the new category of feedback-related issues, but also new types within existing categories. While prior labeling-related issues addressed the presentation of different kinds of static information, the new "Changed state description" looks into dynamic components whose textual description should change based on the state they are currently in. The new navigation-related issue "Unfamiliar layout" refers to accessibility issues caused by inconsistent placement of common UI elements within and across apps. While sighted users can scan the screen to locate such a component, this issue forces screen reader users to navigate the whole screen component by component. Lastly, the issue "Faraway label" occurs when the label for an input field is not located in close proximity, causing confusion about the intent of the input.

4.3.2 Coalescing issues. In comparing issues as defined and categorized in our SLR and how the issues were encountered by users, we observed that multiple issue types often merged with each other to create larger usability hindrances. These combinations happened in multiple granularity levels: on an individual UI component, on adjacent components or across the whole screen. Understanding these instances can help better design accessible components and screens, and develop solution techniques.

An example on **individual granularity** would be the inaccessible 'add Card' button in A_{ak}, which required long press for activation and provided no feedback when the submenu popped up. To the user, these two issues coalesced into an inactionable element.

To exemplify **adjacent granularity**, the combination of unfocusable headers and unlabeled buttons in A_{uh}'s Habit list, resulted in a meaningless table. Even with OCR's help to label the buttons and interviewer hints, the lack of headers still kept the buttons unusable. Sometimes components with no issues by themselves can become inaccessible by adjacent components. When adding a new Card in A_{ak}, despite label and hint present, the edit fields were regarded as inadequately annotated because of unlabeled or mislabeled buttons surrounding them.

Lastly, for **screen-wide granularity**, we observed how, in A_{ak}, the unnatural focus order starting from the edit field instead of the first element on screen and lack of feedback upon saving a new Card, caused users to believe there was an ineffective action.

Takeaway 3

To users, multiple types of accessibility issues can be perceived as one singular usability problem.

4.3.3 Severe issues. As defined in Table 13, we observed the impact of different issues and how testers perceived them, to understand the severity of different category, type and combination of issues. In Table 10, we list the three most severe impacts, defined as follows:

(1) **Blockers**: User was *functionally* unable to proceed as the screen reader failed to reach or activate an element.
(2) **Obstruction - blocking**: While functionally possible, user was *practically* unable to proceed due to inaccessible elements, perceiving them as blockers. They required hints from the interviewer to go forward.
(3) **Obstruction - misleading**: User misunderstood and incorrectly used a functionality.

Interestingly, the most commonly addressed issue, missing labels ($I_{lb}1$), appeared to be the least severe. Its instances in our severe impact list come in the form of specific modalities like tab headers and tables. A primary reason for this reduced impact was the inclusion of auto-generated labels, indicating the effectiveness of *label generating interventions* (Section 3.3.3). However, in most cases, they were reportedly incorrect and users preferred developer-written labels. On the other hand, subjective issues, like the quality of labels or intuitiveness of layout, caused severe hindrances, further diminishing the effectiveness of *programmatic rules* as an oracle for automated detection (Section 3.5.1).

Lastly, we saw that customizing views or using `WebViews` as containers caused the most severe cases of blockers. These did not expose interaction capabilities ($I_{ac}1$) or even focus ($I_{nv}1$) of contained elements to screen readers effectively, rendering the features, or even the app, completely unusable.

Takeaway 4

Most severely impactful accessibility issues are caused by inappropriate use of custom views, and the lack of proper labeling and feedback.

5 Discussion

5.1 User-aware Issue Categorization

To fulfill the primary goal of this paper, to create a user-centered categorization of accessibility issues on mobile for screen reader users, we synthesized our findings from the systematic literature review (SLR) and user study. The SLR contributed to our computational understanding of issues, i.e., what computational reasons cause issues, whether it is a missing attribute in the code or an inappropriate implementation of a UI component. The user study demonstrated the issues' experiential outcome, i.e., how the issues manifested to the users, and impacted their understanding and use of the app. For our categorization, we triangulated these aspects with WCAG, the foundational guidelines for web accessibility.

We adopted WCAG 2.2's four principles as the top-level categories: Perceivable, Operable, Understandable and Robust. Then, based on the impact and perception by users, we placed the reported accessibility issues as a guideline to the appropriate category. Lastly, we incorporated the relevant issue type from Tables 3 and 9 to each guideline as a more technical reference to the issue. In Table 11, we present MCAG (**M**obile **C**ontent **A**ccessibility **G**uidelines) for

Table 9: New accessibility issue types observed in the user study, including a new category for feedback-related issues

ID	Issue Type	Description	Example
Labeling-related issues			
$I_{lb}8$	Changed state description	No change in alternative text if interactable widget changed its visual after user action.	The bookmark button has two states: bookmarked and not bookmarked, conveyed only through icon change.
Navigation-related issues			
$I_{nv}7$	Unfamiliar layout	When the location of commonly used or important elements are not placed in a consistent location.	Buttons for form submission are usually either in the top bar or right after the last edit field.
$I_{nv}8$	Faraway label	Label for an input field is not located in adjacent to the field.	Multiple control buttons are located between the label and the input field.
Feedback-related issues			
$I_{fd}1$	No action feedback	No textual feedback is provided to convey the success or failure of an action.	After submitting a form, there is no "Artifact created successfully" message.
$I_{fd}2$	Inadequate instruction	No instructions are provided to convey the purpose or requirement to complete a task.	Empty dropdown menu does not provide information about the requirement to add an entry first.
$I_{fd}3$	Inadequate progress indication	For dynamically operable elements, no information is provided to indicate progress.	Buttons that require double tap and press do not provide meaningful auditory indication whether the button is operating.

screen reader users. We also publish the guidelines as a working documentation [13] for developers to reference.

5.2 Implications and Future Directions

Based on our findings and takeaways discussed in prior sections, our paper presents the following implications and future directions for mobile accessibility research:

- **Automation for subjective accessibility issues:** Studies so far have been proficient in addressing simple heuristics-based issues, for instance, checking the presence of values in `contentDescription` or `hint` attributes to flag unlabeled elements. During the user studies, we observed that most issues are caused due to subjective challenges: whether a label is correct or informative, whether a focus order or placement of a button is intuitive, etc. Future work needs to focus on addressing these issues for a more comprehensive coverage of user-experienced accessibility issues. LLMs, already showing promise in the field of software accessibility [24, 46, 48, 111, 117], can prove useful in this regard.
- **Feedback-related issues:** We observed a new category of issues from our user studies not mentioned or addressed previously in automated approaches: feedback-related issues. We characterized and categorized them into specific issue types: "no action feedback", "inadequate instruction", and "inadequate progress indication". We also discussed how these issues rank among the most severely impactful accessibility hindrances. Future work can investigate the depth and breadth of this category of issues through further empirical and user studies. Additionally, adopting the newer intervention trends like dynamic input modalities and assisted proxies, as discussed in Section 3.5.1, can lead to automated ways for detecting and fixing these issues.
- **Alternative interactions:** Participants in the user study commonly suggested alternative modes of presenting information or operating interactive elements. For instance, for graphs and charts, users recommended tabular representations. For components that require complex touch-based operations, users suggested alternative text entry-based input mechanisms. Multiple studies have looked into alternative interaction modes for commonly inaccessible components, like charts [32], maps [87], documents [58], and more. Automated resolution strategies by constructing alternative interactions at runtime show potential as future directions.
- **Alternative feedback mechanism:** Another interesting observation from the user studies has been users' preference for non-textual auditory or haptic feedback. For instance, despite the lack of type specification for interactive elements, users were able to determine their actionability based on the specific sound made when focusing on that element. Prior studies have looked into earcons [20], tactile feedback [21, 35], and multimodal gestures [51], as alternative modes beyond text announcements. Further research can utilize this mechanism for addressing issues related to labeling, dynamic change and feedback.
- **Use and further enhancement of MCAG**: The categories and types of issues listed in the MCAG are shaped primarily by accessibility issues addressed by automated interventions, which are reportedly unable to detect the full range of user-experienced issues. While we aimed to augment that limitation with a comprehensive review of the literature and accessibility tests with real screen reader users, further user-first studies and triangulation with different accessibility guidelines can enhance the depth and breadth of the issue categories and types listed on MCAG. For now, with MCAG's current state [13], it works as a resource for learning about common screen reader accessibility issues in mobile and discovering existing tools to detect and fix them. It also paves the way towards establishing an authoritative guideline that takes into account user experiences coupled with computational specifications.

Table 10: List of severe impacts and the issues that caused them in the user study.

Impact	Issue Category	Issue Scenario/Description	UI Modality
Blocker	Activation	Interactive element does not work, or work as expected ($I_{ac}1$).	Customized view, clickable span
	Navigation	Interactive element cannot be focused on ($I_{nv}1$).	Element in WebView
Obstruction - blocking	Activation, Feedback	Button requires non-default interaction (e.g., long press) but does not convey such requirement ($I_{fd}3$).	Interactive element (e.g., button)
	Mislabeling	Button has inadequate description and no interaction type to convey it needs to be interacted with ($I_{lb}5$, $I_{lb}8$).	Interactive element, customized view
		Edit field have insufficient hints without associated labels ($I_{lb}2$, $I_{lb}5$).	Edit field
	Missing label	The tab headers are unlabeled, providing no information which page it leads to ($I_{lb}1$).	Tab headers
	Feedback	Incorrect warning is announced only through screen readers, preventing user from progressing ($I_{fd}2$).	Edit field
Obstruction - misleading	Mislabeling	Empty dropdown does not provide adequate description for its state ($I_{lb}5$).	Custom interactive element
		Adjacent edit fields contain duplicate and ambiguous hints ($I_{lb}6$).	Edit fields
		Button has incorrect label ($I_{lb}5$).	Interactive element
	Navigation, Mislabeling, Missing label	Buttons with incorrect or no labels in a table format with unfocusable headers do not provide user with any context to use elements ($I_{lb}1$, $I_{lb}5$, $I_{nv}1$).	Interactive elements, Table view
	Navigation	User misses out on important information if text view is not focusable ($I_{nv}1$).	Text
		Unfamiliar placement of commonly used buttons leads to incorrect user action ($I_{nv}7$).	Interactive element
	Feedback	No feedback on action causes user to misunderstand state change ($I_{fd}1$).	Interactive element
	Activation	Element intended for visual presentation has interactable features ($I_{ac}3$).	Custom view (e.g., Slider)

6 Threats to Validity

In this section, we describe the possible threats to validity and reliability in our two studies, and our efforts towards mitigation.

6.1 RQ1 - Systematic Literature Review (SLR)

Internal Validity: To reduce sampling bias and avoid missing relevant papers, we analyzed prior SLRs to determine not only a thorough search string but also the most widely used databases. We also conducted forward snowballing, to cover papers that might not have our selected keywords in their titles and abstracts. With sub-research questions defined early on, we were able to narrow down our scope using inclusion/exclusion criteria. Finally, to mitigate reviewer biases during the screening process, two authors individually read the papers and excluded only through consensus.

External Validity: To maximize generalizability, we searched from five different databases and did not impose any limitation on publication venue. We chose 2010 as the earliest cut-off point since both VoiceOver and Talkback were released as early as 2009. The earliest relevant paper we found after conducting the SLR was from 2017.

6.2 RQ2 - User study

Internal Validity: To increase reliability of our user study, we tried mitigating researcher bias, reactivity and respondent bias [62]. To reduce researcher bias, only one author conducted the interviews so that the other was not biased during the analysis process. To address reactivity, our protocol partially facilitated free user navigation without being biased by tasks. Lastly, for respondent bias, we clarified during introduction the nature of the test sessions and how these were apps not developed by us, reducing risk of appeasement. We also followed Robson's mitigation guidelines [85] *prolonged involvement* with the recordings, *triangulation* with the papers in our SLR, and an *audit trail* throughout the interviews and analysis, noting down memos.

External Validity: The generalizability of our findings can be threatened due to the geographic scope of participants (North America), choice of platform (Android), number of apps (4) and the number of issues therein. Participants were mostly recruited through Fable, which is a Canadian organization housing North American testers. While the sociocultural and geographic implications have

Table 11: Mobile Content Accessibility Guidelines (MCAG) for screen reader users, listing 11 guidelines under 4 principles, referring issue types categorized in Table 3.

Principle	Guideline	Description	Ref. Issues
Perceivable Screen reader users must be able to know the information important to the app.	Content Label	All non-decorative content on screen must provide an alternative label that describes, effectively, the presented (image) or expected (edit field) content.	$I_{lb}1$, $I_{lb}2$, $I_{lb}3$, $I_{lb}4$, $I_{lb}5$, $I_{lb}6$
	State Description	Content with multiple states must provide description about their current or changed state.	$I_{dy}5$, $I_{lb}8$
	Focusable	All non-decorative content on screen must be focusable through swipe navigation on screen readers.	$I_{nv}1$, $I_{nv}2$
Operable Screen reader users must be able to activate and understand the activation of interactions offered by the app.	Actionable	All interactive elements can be activated by double-tap.	$I_{ac}1$, $I_{ac}2$
	Action feedback	The success or failure of an action must be properly announced.	$I_{fd}1$
	Alternative Interaction	Interactions dependent on mechanism other than double-tap must be properly conveyed before and/or during action.	$I_{fd}3$, $I_{dy}5$
Understandable Screen reader users must be able to intuitively understand the operations and layout of the screen.	Content Purpose	All interactive content must convey their operational purpose and requirements.	$I_{lb}5$, $I_{lb}7$, $I_{lb}8$, $I_{nv}8$, $I_{fd}2$
	Content Location	The content must be located, through directional focus order, in an intuitive manner.	$I_{nv}3$, $I_{nv}4$, $I_{nv}5$, $I_{nv}7$, $I_{nv}8$
	Content Relocation	If content is removed, added or moved in the screen, the relocation information must be conveyed to the user.	$I_{dy}1$, $I_{dy}2$, $I_{dy}3$, $I_{dy}4$
Robust Screen reader users must be able to interpret all important content via screen reader.	Compatible	The content must be compatible to screen readers, despite its underlying implementation.	$I_{nv}1$, $I_{ac}1$, $I_{ac}2$
	Consistent	Content inaccessible to non-assisted user must not be accessible through screen reader.	$I_{ac}3$, $I_{nv}1$, $I_{nv}6$

not been explored, the global reach of Android devices (72.5-73.9% of market share [67]) support the universality of our findings.

The decision to select Android apps has been informed by the majority of prior studies developing interventions for Android, owing to the access to open source repositories. That also gave us the opportunity to inject issues.

While there can always be the case of including more apps, we decided on four. We followed the standards from related work that also conducted user studies on four apps in conjunction with empirical evaluations [70, 91].

To incorporate as many accessibility issue types as possible, we selected apps with existing issues and updated the code in their open source repository to inject more issues. Despite our efforts, not all possible accessibility issues have been evaluated. Therefore, the categorization is presented not as a final version, but as a starting point towards a more robust guideline.

7 Conclusion

In this paper, we conducted a mixed-methods study, consisting of a Systematic Literature Review (SLR) and a user study, to categorize accessibility issues in mobile for screen reader users. From the SLR, we analyzed four types of automated testing and fixing interventions to generate an initial set of issue categories. We complemented this list with actual user experience, through 20 user studies on Android apps with blind individuals. These studies demonstrated a current trend towards high-fidelity automation to detect issues and annotate solutions, while revealing a significant limitation in current interventions to address subjective and feedback-related accessibility issues, commonly occurring in user studies. Finally, we constructed MCAG, a pilot issue categorization based on WCAG, focused on the mobile platform for screen reader users. The goal of MCAG is to pioneer the construction of a more authoritative guideline for accessible mobile app development.

Acknowledgments

This work has been supported by award numbers 2211790 and 2106306 from the National Science Foundation. We thank Fable for their collaboration in facilitating accessibility tests. We are grateful for the feedback from the anonymous reviewers, which helped improve this work.

References

[1] Hawraa Abdulreda, Iyad Abu Doush, and Zainab AlMeraj. 2024. E-government Mobile Web Accessibility Challenges facing People with Visual Impairments: A Multi-Method Evaluation. In *Proceedings of the 11th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. 124–132.

[2] Patricia Acosta-Vargas, Belén Salvador-Acosta, Luis Salvador-Ullauri, William Villegas-Ch, and Mario Gonzalez. 2021. Accessibility in native mobile applications for users with disabilities: A scoping review. *Applied Sciences* 11, 12 (2021), 5707.

[3] Muna Al-Razgan, Sarah Almoaiqel, Nuha Alrajhi, Alyah Alhumegani, Abeer Alshehri, Bashayr Alnefaie, Raghad AlKhamiss, and Shahad Rushdi. 2021. A systematic literature review on the usability of mobile applications for visually impaired users. *PeerJ Computer Science* 7 (2021), e771.

[4] Nancy Alajarmeh. 2021. Non-visual access to mobile devices: A survey of touchscreen accessibility for users who are visually impaired. *Displays* 70 (2021), 102081.

[5] Ali S Alotaibi, Paul T Chiou, and William GJ Halfond. 2022. Automated detection of talkback interactive accessibility failures in android applications. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE Computer Society, 232–243.

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

[7] Android. 2025. Build accessible apps. https://developer.android.com/guide/topics/ui/accessibility/. Accessed: 2025-08-20.

[8] Android. 2025. Espresso,. https://developer.android.com/training/testing/espresso/. Accessed: 2025-09-08.

[9] Android. 2025. Improve your code with lint checks,. https://developer.android.com/studio/write/lint/. Accessed: 2025-09-08.

[10] Android. 2025. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/other-testing-tools/monkey. Accessed: 2025-09-08.

[11] Android. 2025. Write automated tests with UI Automator. https://developer.android.com/training/testing/other-components/ui-automator. Accessed: 2025-09-08.

[12] ankidroid. 2025. *AnkiDroid*. Retrieved September 1, 2025 from https://github.com/ankidroid/Anki-Android

[13] Anon. 2025. MCAG - Mobile Content Accessibility Guidelines. https://mcag2026.github.io/mcag2026/. Accessed: 2025-01-12.

[14] Apple. 2025. Accessibility | Apple Developer Documentation. https://developer.apple.com/design/human-interface-guidelines/accessibility/. Accessed: 2025-08-20.

[15] Apple. 2025. Accessibility Inspector. https://developer.apple.com/documentation/accessibility/accessibility-inspector/. Accessed: 2025-09-08.

[16] Association for Computing Machinery. 2025. ACM Digital Library. https://dl.acm.org/. Accessed: 2025-09-02.

[17] Mars Ballantyne, Archit Jha, Anna Jacobsen, J Scott Hawker, and Yasmine N El-Glaly. 2018. Study of accessibility guidelines of mobile applications. In *Proceedings of the 17th international conference on mobile and ubiquitous multimedia*. 305–315.

[18] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology* 3, 2 (2006), 77–101. arXiv:https://doi.org/10.1191/1478088706qp063oa doi:10.1191/1478088706qp063oa

[19] Buildfire. 2025. 10 App Categories That Will Dominate 2025 and Beyond. https://buildfire.com/app-categories/. Accessed: 2025-01-12.

[20] Isna Alfi Bustoni, Indra Hidayatulloh, Azhari SN, and Nia Gella Augoestin. 2018. Multidimensional Earcon Interaction Design for The Blind: a Proposal and Evaluation. In *2018 International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. 384–388. doi:10.1109/ISRITI.2018.8864487

[21] Maria Claudia Buzzi, Marina Buzzi, Barbara Leporini, and Maria Teresa Paratore. 2013. Vibro-tactile enrichment improves blind user interaction with mobile touchscreens. In *IFIP Conference on Human-Computer Interaction*. Springer, 641–648.

[22] Volkan Çalışkan, Özgürol Öztürk, and Kerem Rızvanoğlu. 2014. Mobile Accessibility in Touchscreen Devices: Implications from a Pilot Study with Blind Users on iOS Applications in iPhone and iPad. In *Research and Design Innovations for Mobile User Experience*. IGI Global Scientific Publishing, 182–202.

[23] Michael Crystian Nepomuceno Carvalho, Felipe Silva Dias, Aline Grazielle Silva Reis, and André Pimenta Freire. 2018. Accessibility and usability problems encountered on websites and applications in mobile devices by blind and normal-vision users. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (Pau, France) *(SAC '18)*. Association for Computing Machinery, New York, NY, USA, 2022–2029. doi:10.1145/3167132.3167349

[24] Jieshan Chen, Chunyang Chen, Zhenchang Xing, Xiwei Xu, Liming Zhu, Guoqiang Li, and Jinshui Wang. 2020. Unblind your apps: predicting natural-language labels for mobile GUI components by deep learning. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (Seoul, South Korea) *(ICSE '20)*. Association for Computing Machinery, New York, NY, USA, 322–334. doi:10.1145/3377811.3380327

[25] Jieshan Chen, Amanda Swearngin, Jason Wu, Titus Barik, Jeffrey Nichols, and Xiaoyi Zhang. 2022. Towards complete icon labeling in mobile applications. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–14.

[26] Sen Chen, Chunyang Chen, Lingling Fan, Mingming Fan, Xian Zhan, and Yang Liu. 2021. Accessible or not? an empirical investigation of android app accessibility. *IEEE Transactions on Software Engineering* 48, 10 (2021), 3954–3968.

[27] Clarivate. 2025. Web of Science. https://www.webofscience.com/wos/woscc/smart-search. Accessed: 2025-09-02.

[28] Mauricio Cruz-Portilla, Juan Carlos Pérez-Arriaga, Jorge Octavio Ocharán-Hernández, and Ángel J Sánchez-García. 2021. Accessibility in the software development life cycle: a systematic literature review. In *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 97–103.

[29] Daniela S Cruzes and Tore Dyba. 2011. Recommended steps for thematic synthesis in software engineering. In *2011 international symposium on empirical software engineering and measurement*. IEEE, 275–284.

[30] Cícero Joasyo Mateus De Moura, Souza De Oliveira, Kenyo Abadio Crosara Faria, and Eduardo Noronha de Andrade Freitas. 2017. A new API for android accessibility testing. In *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 594–598.

[31] Camila Dias de Oliveira, Maria Lydia Fioravanti, Renata Pontin de Mattos Fortes, and Ellen Francine Barbosa. 2018. Accessibility in mobile applications for elderly users: a systematic mapping. In *2018 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–9.

[32] Cynthya Letícia Teles de Oliveira, Alan Trindade de Almeida Silva, Jefferson Magalhães de Morais, and Marcelle Pereira Mota. 2020. ChartVision: accessible vertical bar charts. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems*. 1–10.

[33] Gabriela AA de Oliveira, Otavio de Faria Oliveira, Stenio de Abreu, Raphael W de Bettio, and André P Freire. 2022. Opportunities and accessibility challenges for open-source general-purpose home automation mobile applications for visually disabled users. *Multimedia Tools and Applications* 81, 8 (2022), 10695–10722.

[34] Digital Scholar. 2025. Zotero - Your personal research assistant. https://www.zotero.org. Accessed: 2025-09-02.

[35] Yasmine N El-Glaly, Francis Quek, Tonya Smith-Jackson, and Gurjot Dhillon. 2013. Touch-screens are not tangible: Fusing tangible interaction with touch glass in readers for the blind. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*. 245–253.

[36] 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.

[37] Elsevier. 2025. Science Direct. https://www.sciencedirect.com/. Accessed: 2025-09-02.

[38] Elsevier. 2025. Scopus. https://www.scopus.com/. Accessed: 2025-09-02.

[39] Fable. 2025. The only accessibility platform powered by people with disabilities. https://makeitfable.com/

[40] Andre Pimenta Freire, Rudinei Goularte, and Renata Pontin de Mattos Fortes. 2007. Techniques for developing more accessible web applications: a survey towards a process classification. In *Proceedings of the 25th Annual ACM international Conference on Design of Communication*. 162–169.

[41] Anderson Canale Garcia, Silvana Maria Affonso de Lara, Lianna Mara Castro Duarte, Renata Pontin de Mattos Fortes, and Kamila Rios Da Hora Rodrigues. 2023. Early accessibility testing–an automated kit for Android developers. In *Proceedings of the 29th Brazilian Symposium on Multimedia and the Web*. 11–15.

[42] Google. 2025. Control your Android device with Switch Access. https://support.google.com/accessibility/android/answer/6122836?hl=en. Accessed: 2025-09-08.

[43] Google. 2025. Get started with Accessibility Scanner,. https://support.google.com/accessibility/android/answer/6376570?hl=en/. Accessed: 2025-09-08.

[44] Ming Gu, Lei Pei, Sheng Zhou, Ming Shen, Yuxuan Wu, Zirui Gao, Ziwei Wang, Shuo Shan, Wei Jiang, Yong Li, and Jiajun Bu. 2025. Towards an Inclusive Mobile Web: A Dataset and Framework for Focusability in UI Accessibility. In *Proceedings of the ACM on Web Conference 2025* (Sydney NSW, Australia) *(WWW '25)*. Association for Computing Machinery, New York, NY, USA, 5096–5107. doi:10.1145/3696410.3714523

[45] Ziyao He, Syed Fatiul Huq, and Sam Malek. 2024. " I tend to view ads almost like a pestilence": On the Accessibility Implications of Mobile Ads for Blind Users. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. 1–13.

[46] Ziyao He, Syed Fatiul Huq, and Sam Malek. 2025. Enhancing Web Accessibility: Automated Detection of Issues with Generative AI. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE101 (June 2025), 24 pages. doi:10.1145/3729371

[47] Syed Fatiul Huq, Abdulaziz Alshayban, Ziyao He, and Sam Malek. 2023. #A11yDev: Understanding Contemporary Software Accessibility Practices from Twitter Conversations. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 217, 18 pages. doi:10.1145/3544548.3581455

[48] Syed Fatiul Huq, Mahan Tafreshipour, Kate Kalcevich, and Sam Malek. 2024. Automated Generation of Accessibility Test Reports from Recorded User Transcripts. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 534–546.

[49] IEEE. 2025. IEEE Xplore. https://ieeexplore.ieee.org/search/advanced. Accessed: 2025-09-02.

[50] International Agency for the Prevention of Blindness. 2020. Vision loss could be treated in one billion people worldwide. https://www.iapb.org/news/the-lancet-global-health-vision-loss-could-be-treated-in-one-billion-people-worldwide/. Accessed: 2025-08-19.

[51] Nor Azman Ismail, Yohgamalar Naidu Gunasegaran, Layla Hasan, Fuad Abdulgaleel Abdoh Ghaleb, and Yee Yong Pang. 2025. Designing and usability evaluation of multimodal input modalities for visually impaired persons on mobile platforms. *Universal Access in the Information Society* (2025), 1–13.

[52] iSoron. 2025. *Loop Habit Tracker*. Retrieved September 1, 2025 from https://github.com/iSoron/uhabits

[53] itkach. 2024. *Aard 2 for Android App*. Retrieved September 1, 2025 from https://github.com/itkach/aard2-android

[54] Barbara Kitchenham, Stuart Charters, et al. 2007. Guidelines for performing systematic literature reviews in software engineering. (2007).

[55] Yekaterina Kosova and Milera Izetova. 2020. Accessibility of mathematics MOOCs to learners with disabilities. 1 (eng) (2020), 205–229.

[56] Bimal Aklesh Kumar and Priya Mohite. 2018. Usability of mobile learning applications: a systematic literature review. *Journal of Computers in Education* 5, 1 (2018), 1–17.

[57] Barbara Leporini, Marina Buzzi, and Marion Hersh. 2023. Video conferencing tools: Comparative study of the experiences of screen reader users and the development of more inclusive design guidelines. *ACM Transactions on Accessible Computing* 16, 1 (2023), 1–36.

[58] Barbara Leporini and Giuseppe Catanzaro. 2020. App inventor as a developing tool to increase the accessibility and readability of information: A case study. In *Proceedings of the 9th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. 71–75.

[59] Barbara Leporini and Eleonora Palmucci. 2017. A mobile educational game accessible to all, including screen reading users on a touch-screen device. In *Proceedings of the 16th World Conference on Mobile and Contextual Learning*. 1–4.

[60] Barbara Leporini and Eleonora Palmucci. 2018. Accessible question types on a touch-screen device: the case of a mobile game app for blind people. In *International conference on computers helping people with special needs*. Springer, 262–269.

[61] Yang Li, Gang Li, Luheng He, Jingjie Zheng, Hong Li, and Zhiwei Guan. 2020. Widget Captioning: Generating Natural Language Description for Mobile User Interface Elements. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 5495–5510. doi:10.18653/v1/2020.emnlp-main.443

[62] Yvonna S Lincoln and Egon G Guba. 1985. *Naturalistic inquiry*. sage.

[63] Zhe Liu, Chunyang Chen, Junjie Wang, Mengzhuo Chen, Boyu Wu, Yuekai Huang, Jun Hu, and Qing Wang. 2024. Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 51, 20 pages. doi:10.1145/3613904.3642939

[64] Renan Lopes, Agebson Rocha Façanha, and Windson Viana. 2022. I can't pay! Accessibility analysis of mobile banking apps. In *Proceedings of the Brazilian Symposium on Multimedia and the Web*. 253–257.

[65] Ying Ma, Chuyi Yu, Ming Yan, Arun Kumar Sangaiah, and Youke Wu. 2023. Dark-side avoidance of mobile applications with data biases elimination in socio-cyber world. *IEEE Transactions on Computational Social Systems* 11, 4 (2023), 4955–4964.

[66] Stephane Madeira, Frederico Branco, Ramiro Gonçalves, Manuel Au-Yong-Oliveira, Fernando Moreira, and José Martins. 2021. Accessibility of mobile applications for tourism—is equal access a reality? *Universal Access in the Information Society* 20, 3 (2021), 555–571.

[67] SQ Magazine. 2025. Android Statistics 2025: Key Data on Devices, Apps & AI Trends. https://sqmagazine.co.uk/android-statistics. Accessed: 2025-01-12.

[68] Siti Ummi Masruroh, Nanda Alivia Rizqy Vitalaya, Husni Teja Sukmana, Imam Subchi, Dewi Khairani, and Yusuf Durachman. 2022. Evaluation of usability and accessibility of mobile application for people with disability: Systematic literature review. In *2022 International Conference on Science and Technology (ICOSTECH)*. IEEE, 1–7.

[69] Delvani Antônio Mateus, Carlos Alberto Silva, Arthur FBA De Oliveira, Heitor Costa, and André Pimenta Freire. 2021. A systematic mapping of accessibility problems encountered on websites and mobile apps: A comparison between automated tests, manual inspections and user evaluations. *Journal on Interactive Systems* 12, 1 (2021), 145–171.

[70] Delvani Antônio Mateus, Carlos Alberto Silva, Marcelo Medeiros Eler, and André Pimenta Freire. 2020. Accessibility of mobile applications: evaluation by users with visual impairment and by automated tools. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems* (Diamantina, Brazil) *(IHC '20)*. Association for Computing Machinery, New York, NY, USA, Article 4, 10 pages. doi:10.1145/3424953.3426633

[71] Mark McKay. 2017. Accessibility challenges of hybrid mobile applications. In *International Conference on Universal Access in Human-Computer Interaction*. Springer, 193–208.

[72] Forough Mehralian, Ziyao He, and Sam Malek. 2024. Automated Accessibility Analysis of Dynamic Content Changes on Mobile Apps. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 482–494.

[73] Forough Mehralian, Navid Salehnamadi, Syed Fatiul Huq, and Sam Malek. 2022. Too much accessibility is harmful! automated detection and analysis of overly accessible elements in mobile apps. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–13.

[74] Forough Mehralian, Navid Salehnamadi, and Sam Malek. 2021. Data-driven accessibility repair revisited: on the effectiveness of generating labels for icons in Android apps. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Athens, Greece) *(ESEC/FSE 2021)*. Association for Computing Machinery, New York, NY, USA, 107–118. doi:10.1145/3468264.3468604

[75] Nada Nasser. 2025. *Money Manager App*. Retrieved September 1, 2025 from https://github.com/Nada-Nasser/Money-Manager-App

[76] Tanja Nedovic, Nesljihan Umeri-Sali, and Kerstin Denecke. 2019. Supporting blind and visually impaired persons in managing their medication. In *German Medical Data Sciences: Shaping Change–Creative Solutions for Innovative Medicine*. IOS Press, 189–196.

[77] offa. 2025. A list of Free and Open Source Software (FOSS). https://github.com/offa/android-foss. Accessed: 2025-01-12.

[78] Alberto Dumont Alves Oliveira, Paulo Sergio Henrique dos Santos, Wajdi Aljedaani, and Marcelo Medeiros Eler. 2024. Exploring the influence of software evolution on mobile app accessibility: Insights from user reviews. *Journal of the Brazilian Computer Society* 30, 1 (2024), 584–607.

[79] Karla Ordoñez, José Hilera, and Samanta Cueva. 2022. Model-driven development of accessible software: a systematic literature review. *Universal Access in the Information Society* 21, 1 (2022), 295–324.

[80] Matthew J Page, Joanne E McKenzie, Patrick M Bossuyt, Isabelle Boutron, Tammy C Hoffmann, Cynthia D Mulrow, Larissa Shamseer, Jennifer M Tetzlaff, Elie A Akl, Sue E Brennan, Roger Chou, Julie Glanville, Jeremy M Grimshaw, Asbjørn Hróbjartsson, Manoj M Lalu, Tianjing Li, Elizabeth W Loder, Evan Mayo-Wilson, Steve McDonald, Luke A McGuinness, Lesley A Stewart, James Thomas, Andrea C Tricco, Vivian A Welch, Penny Whiting, and David Moher. 2021. The PRISMA 2020 statement: an updated guideline for reporting systematic reviews. *BMJ* 372 (2021). arXiv:https://www.bmj.com/content/372/bmj.n71.full.pdf doi:10.1136/bmj.n71 Publisher: BMJ Publishing Group Ltd..

[81] Eunju Park, Sungjun Han, Hogon Bae, Raekyung Kim, Seungjae Lee, Daejune Lim, and Hankyu Lim. 2019. Development of automatic evaluation tool for mobile accessibility for android application. In *2019 International Conference on Systems of Collaboration Big Data, Internet of Things & Security (SysCoBIoTS)*. IEEE, 1–6.

[82] pcqpcq. 2025. Open Source Android Apps. https://github.com/pcqpcq/open-source-android-apps. Accessed: 2025-01-12.

[83] Program-L. 2025. Program-L: VI Programmers Discussion List. https://www.freelists.org/list/program-l. Discussion group for visually impaired computer programmers.

[84] Nataša Rajh, Klaus Miesenberger, and Reinhard Koutny. 2024. Accessibility in the software engineering (SE) process and in integrated development environments (IDEs): a systematic literature review. In *International Conference on Computers Helping People with Special Needs*. Springer, 11–18.

[85] Colin Robson. 2002. *Real world research: A resource for social scientists and practitioner-researchers*. Wiley-Blackwell.

[86] Anne Spencer Ross, Xiaoyi Zhang, James Fogarty, and Jacob O. Wobbrock. 2020. An Epidemiology-inspired Large-scale Analysis of Android App Accessibility. *ACM Trans. Access. Comput.* 13, 1, Article 4 (April 2020), 36 pages. doi:10.1145/3348797

[87] Sebastian Rottmann, Claudia Loitsch, and Gerhard Weber. 2022. Accessible mobile map application and interaction for people with visual or mobility impairments. In *Proceedings of Mensch und Computer 2022*. 119–127.

[88] Navid Salehnamadi, Abdulaziz Alshayban, Jun-Wei Lin, Iftekhar Ahmed, Stacy Branham, and Sam Malek. 2021. Latte: Use-case and assistive-service driven automated accessibility testing framework for android. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–11.

[89] Navid Salehnamadi, Ziyao He, and Sam Malek. 2023. Assistive-technology aided manual accessibility testing in mobile apps, powered by record-and-replay. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–20.

[90] Navid Salehnamadi, Forough Mehralian, and Sam Malek. 2022. Groundhog: An automated accessibility crawler for mobile apps. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. 1–12.
[91] Letícia Seixas Pereira, Maria Matos, and Carlos Duarte. 2024. Exploring Mobile Device Accessibility: Challenges, Insights, and Recommendations for Evaluation Methodologies. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 964, 17 pages. doi:10.1145/3613904.3642526
[92] Imani N Sherman, Jasmine D Bowers, Liz-Laure Laborde, Juan E Gilbert, Jaime Ruiz, and Patrick G Traynor. 2020. Truly visual caller id? an analysis of anti-robocall applications and their accessibility to visually impaired users. In *2020 IEEE International Symposium on Technology and Society (ISTAS)*. IEEE, 266–279.
[93] Clauirton A. Siebra, Walter Correia, Marcelo Penha, Jefté Macêdo, Jonysberg Quintino, Marcelo Anjos, Fabiana Florentin, Fabio Q. B. Silva, and Andre L M Santos. 2018. An analysis on tools for accessibility evaluation in mobile applications. In *Proceedings of the XXXII Brazilian Symposium on Software Engineering* (Sao Carlos, Brazil) *(SBES '18)*. Association for Computing Machinery, New York, NY, USA, 172–177. doi:10.1145/3266237.3266238
[94] Camila Silva, Marcelo Medeiros Eler, and Gordon Fraser. 2018. A survey on the tool support for the automatic evaluation of mobile accessibility. In *Proceedings of the 8th International Conference on Software Development and Technologies for Enhancing Accessibility and Fighting Info-exclusion*. 286–293.
[95] J Silva, Catarina Silva, Luís Marcelino, Rui Ferreira, and António Pereira. 2011. Assistive mobile software for public transportation. *Proc. UBICOMM* (2011).
[96] Simerneet Singh, Nishtha Jatana, Sukriti Sehgal, Rakshita Anand, B Arunkumar, and Janjhyam Venkata Naga Ramesh. 2024. Making digital payments accessible beyond sight: a usability study of UPI-Based smartphone applications. *IEEE Access* 12 (2024), 6830–6841.
[97] Ting Su, Guozhu Meng, Yuting Chen, Ke Wu, Weiming Yang, Yao Yao, Geguang Pu, Yang Liu, and Zhendong Su. [n. d.]. Guided, stochastic model-based GUI testing of Android apps. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*. 245–256.
[98] Amanda Swearngin, Jason Wu, Xiaoyi Zhang, Esteban Gomez, Jen Coughenour, Rachel Stukenborg, Bhavya Garg, Greg Hughes, Adriana Hilliard, Jeffrey P. Bigham, and Jeffrey Nichols. 2024. Towards Automated Accessibility Report Generation for Mobile Apps. *ACM Trans. Comput.-Hum. Interact.* 31, 4, Article 54 (Sept. 2024), 44 pages. doi:10.1145/3674967
[99] Maryam Taeb, Amanda Swearngin, Eldon Schoop, Ruijia Cheng, Yue Jiang, and Jeffrey Nichols. 2024. AXNav: Replaying Accessibility Tests from Natural Language. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 962, 16 pages. doi:10.1145/3613904.3642777
[100] Sachin Tanwar and PVM Rao. 2024. Inequality in User Experience: Can Mobile User Interfaces that Help Sighted Users Create Barriers for Visually Challenged People?. In *International Conference on Computers Helping People with Special Needs*. Springer, 19–30.
[101] Pedro Teixeira, Celeste Eusebio, and Leonor Teixeira. 2024. Understanding the integration of accessibility requirements in the development process of information systems: a systematic literature review. *Requirements Engineering* 29, 2 (2024), 143–176.
[102] Maarten Van Someren, Yvonne F. Barnard, and Jacob Sandberg. 1994. *The Think Aloud Method: A Practical Approach to Modelling Cognitive Processes*. Academic Press, London.
[103] W3C. 2024. Web Content Accessibility Guidelines (WCAG) 2.2. https://www.w3.org/TR/WCAG22/. Accessed: 2025-08-20.
[104] WebAIM. 2022. WebAIM: The WebAIM Million - The 2022 report on the accessibility of the top 1,000,000 home pages. https://webaim.org/projects/million/. (Accessed on 08/24/2022).
[105] WebAIM. 2024. Screen Reader User Survey #10 Results. https://webaim.org/projects/screenreadersurvey10/. Accessed: 2025-08-20.
[106] Brian Wentz and Jonathan Lazar. 2011. Are separate interfaces inherently unequal? An evaluation with blind users of the usability of two interfaces for a social networking platform. In *Proceedings of the 2011 iConference*. 91–97.
[107] Gian Wild. 2018. The inaccessibility of video players. In *International Conference on Computers Helping People with Special Needs*. Springer, 47–51.
[108] World Health Organization. 2023. Blindness and vision impairment. https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment. Accessed: 2025-08-19.
[109] Tao Xin, Jiying Zhu, Luling Wang, and Xiaowei Qin. 2023. Screen Recognition: Creating Accessibility Metadata for Mobile Applications using View Type Detection. In *2023 9th International Conference on Computer and Communications (ICCC)*. IEEE, 1787–1793.
[110] Shunguo Yan and PG Ramachandran. 2019. The current status of accessibility in mobile apps. *ACM Transactions on Accessible Computing (TACCESS)* 12, 1 (2019), 1–31.
[111] Mengxi Zhang, Huaxiao Liu, Changhao Du, Tengmei Wang, Han Li, Pei Huang, and Chunyang Chen. 2025. Distinguishing GUI Component States for Blind Users using Large Language Models. *ACM Trans. Softw. Eng. Methodol.* (March 2025). doi:10.1145/3722106 Just Accepted.
[112] Mengxi Zhang, Huaxiao Liu, Yuheng Zhou, Chunyang Chen, Pei Huang, and Jian Zhao. 2024. Don't Confuse! Redrawing GUI Navigation Flow in Mobile Apps for Visually Impaired Users. *IEEE Transactions on Software Engineering* 50, 12 (Dec 2024), 3351–3368. doi:10.1109/TSE.2024.3485225
[113] Xiaoyi Zhang, Lilian De Greef, Amanda Swearngin, Samuel White, Kyle Murray, Lisa Yu, Qi Shan, Jeffrey Nichols, Jason Wu, Chris Fleizach, et al. 2021. Screen recognition: Creating accessibility metadata for mobile applications from pixels. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–15.
[114] Xiaoyi Zhang, Anne Spencer Ross, Anat Caspi, James Fogarty, and Jacob O Wobbrock. 2017. Interaction proxies for runtime repair and enhancement of mobile application accessibility. In *Proceedings of the 2017 CHI conference on human factors in computing systems*. 6024–6037.
[115] Xiaoyi Zhang, Anne Spencer Ross, and James Fogarty. 2018. Robust annotation of mobile application interfaces in methods for accessibility repair and enhancement. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*. 609–621.
[116] Yuxin Zhang, Sen Chen, Xiaofei Xie, Zibo Liu, and Lingling Fan. 2025. Scenario-Driven and Context-Aware Automated Accessibility Testing for Android Apps. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. IEEE Computer Society, 630–630.
[117] Mingyuan Zhong, Ruolin Chen, Xia Chen, James Fogarty, and Jacob O Wobbrock. 2025. ScreenAudit: Detecting Screen Reader Accessibility Errors in Mobile Apps Using Large Language Models. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*. 1–19.

A RQ1: SLR

A.1 Thematic Synthesis of Automated Interventions

To conduct a systematic analysis of the automated intervention techniques, we followed a related SLR [28] that categorizes accessibility techniques within the software development life cycle and performed thematic synthesis. As prescribed by Cruzes and Dyba [29], we took the following steps:

(1) **Extract data**: Two authors individually read the selected papers, focusing particularly on the sections outlining the techniques' details.
(2) **Code data**: We conducted inductive coding to characterize the techniques, trying to annotate the purpose of the technique, the data modalities they worked with, the type of computation and so on.
(3) **Translate code into themes**: After exchanging individual codes, we merged similar codes and grouped them, leading to categories and themes. At this stage, we also started removing codes or categories that did not contribute to our research question. This included *output modality*, *evaluation method* and so on.
(4) **Create a model of higher order themes**: We mapped themes to each other based on shared code and categories.
(5) **Assess the trustworthiness of the synthesis**: In creating the hierarchy, we ideated on different dimensions to delineate the higher-order themes and decided on purpose of intervention.

In conclusion, we derived two main themes: *test* and *fix*, based on intervention's purpose. Test is further branched based on purpose, with *crawlers* including techniques intended to work as standalone testing tool while *automation support* containing techniques that streamline accessibility development in other processes. On the other hand, fixes are also divided into two sub-themes: those that

resolve image icons only (*label generators*) and those that target any UI component (*UI annotators*). The synthesis map is illustrated in Figure 3.

The resulting synthesis contains 30 codes across 5 categories: input modality, oracle, supported process, prediction model and target component. Among these, input modality is common across all themes, and is repeated in Figure 3 to better display how different techniques are mapped to different sets of input modalities.

A.2 Automated Interventions from the Systematic Literature Review

A.2.1 Automatic Crawlers. In 2019, MATE, developed by P29 [36] was the first automated accessibility crawling tool for Android that detected issues based on ATF. They implemented a random widget-aware exploration strategy with UIAutomator [11] to crawl pages and extract `AccessibilityNodeInfo` for analysis.

Inspired from this technique, P23 [6] conducted a large scale empirical study on the accessibility of Android applications. Keeping ATF as the core oracle, they improved upon MATE by adding more heuristic implementations. They also changed the crawler to Monkey [10], a pseudo-random screen explorer from Android.

Moving away from ATF, P19 [5] developed ATARI, a tool that compares the navigation graphs between Talkback swipe navigation and touchscreen ("non-assisted") navigation. This approach detects UI components that are visible to sighted users but not focusable via TalkBack swipe gestures, the default way blind users navigate apps.

XBot was developed by P18 [26] to conduct a large-scale empirical study of Android accessibility. They also used ATF as an oracle, similar to P29 and P23. However, P18 improved on coverage by analyzing app APKs, extracting intent information, and conducting a more directed crawling.

P16 [73] developed OverSight to detect over-accessibility issues, where elements are focusable or actionable through Talkback but are not visible on the screen for a sighted user. They analyzed both `AccessibilityNodeInfo` and associated screenshots of the screens. Through empirical study of 20 apps, they created their own list of rules as an oracle for detecting overly-focusable and overly-actionable elements.

Groundhog, developed by P15 [90], detects focus and action-related accessibility issues for screen readers and switch devices (e.g., SwitchAccess [42], which lets users with motor disability traverse apps). To increase fidelity to actual user experience, the tool runs multiple proxies — Touch, TalkBack and Abstract Accessibility-API — on a running app. To reduce randomness, they crawled apps using Stoat [97], a stochastic model-based GUI exploration method. Finding disparity in the list of visited and interacted elements between the proxies, Groundhog can detect unfocusable and unactionable elements.

P7 [45] adopted Groundhog's approach to create AdMole, an automated crawler that detects accessibility issues of advertisements externally injected into Android applications.

P3 [72] focused on dynamically changing elements on the screen, such as appearing or disappearing cancel buttons on ads, or button with dynamically updating content, with or without the user's interaction. From an empirical study on 50 apps and a user study with 3 blind testers, they formulated their own rules for dynamic content-change-related accessibility issues. They developed the tool TimeStump, which crawls apps using Stoat, extracts snapshots of changing UI, and based on their rules, determines inaccessible dynamic elements.

P2 [116] improves upon XBot with their context-aware detection tool, A11yScan. They aimed to reduce incorrect detections by adding a static resource tree, from decompiling the APK, to the runtime `AccessibilityNodeInfo` for analysis. They also increased coverage by incorporating not just Activities (Android UI screen) but also other UI presentations (e.g., menus, drawers, dialogs etc.).

Most recently, P1 [117] implemented LLM as an issue detection oracle. Their tool, ScreenAudit, automatically traverses apps using TalkBack and feeds the LLM the TalkBack announcements, along with some contextual information about the app. The generated report was evaluated by accessibility experts.

A.2.2 Automation Support. P31 [30] provided an API for Android that can test for accessibility on Android Classes. The oracle they used for testing was a subset of ATF.

P27 [81] developed a tool that takes in XML files, code of the UI in Android, and checks for missing alternative texts in the UI components.

P24 [86] adopted ATF and added a few of their own experientially determined heuristics, to analyze screenshots and JSON view hierarchies for accessibility issues. While usable as automation support for developers, the purpose of their study was to empirically investigate accessibility issues in Android apps.

P21 [88] developed an automated testing framework, Latte, that can run existing functional unit tests of an Android project using Talkback navigation. Failure of those test cases through Latte can unearth focus and action-related issues latent in the apps.

P14 [41] demonstrated an accessibility toolkit for early testing on Android apps. Through developer studies with students, the paper's primary purpose was to provide easily executable accessibility tests for developers.

P13 [89] proposed A11yPuppetry, a successor to Groundhog that also used TalkBack proxies to determine accessibility from manual testing. Unlike Groundhog, which relied on automated crawlers for coverage, A11yPuppetry converts manual navigation by sighted testers into TalkBack swipe navigation. The paper emphasized on the need of manual testing by developers, and aimed to alleviate the limitation of developers' lack of assistive technology experience. A11yPuppetry provides both automated detection of issues and recordings of TalkBack navigation of developer inference.

While Latte ran functional unit tests through Talkback, AXNav by P12 [99] converts test instructions in natural language forms to accessibility tests on iOS. To detect issues, the project conducted a formative study with QA professionals and constructed their own rules.

P11 [98], another intervention on iOS, focused on creating more succinct and less noisy accessibility reports from existing testing techniques — automated tools or manual audits. As input, the tool takes audit data that annotates UI elements with relevant accessibility issues, and the associated screenshot. They used Apple's Accessibility Inspector [15] as the oracle, the iOS equivalent to Android's ATF.

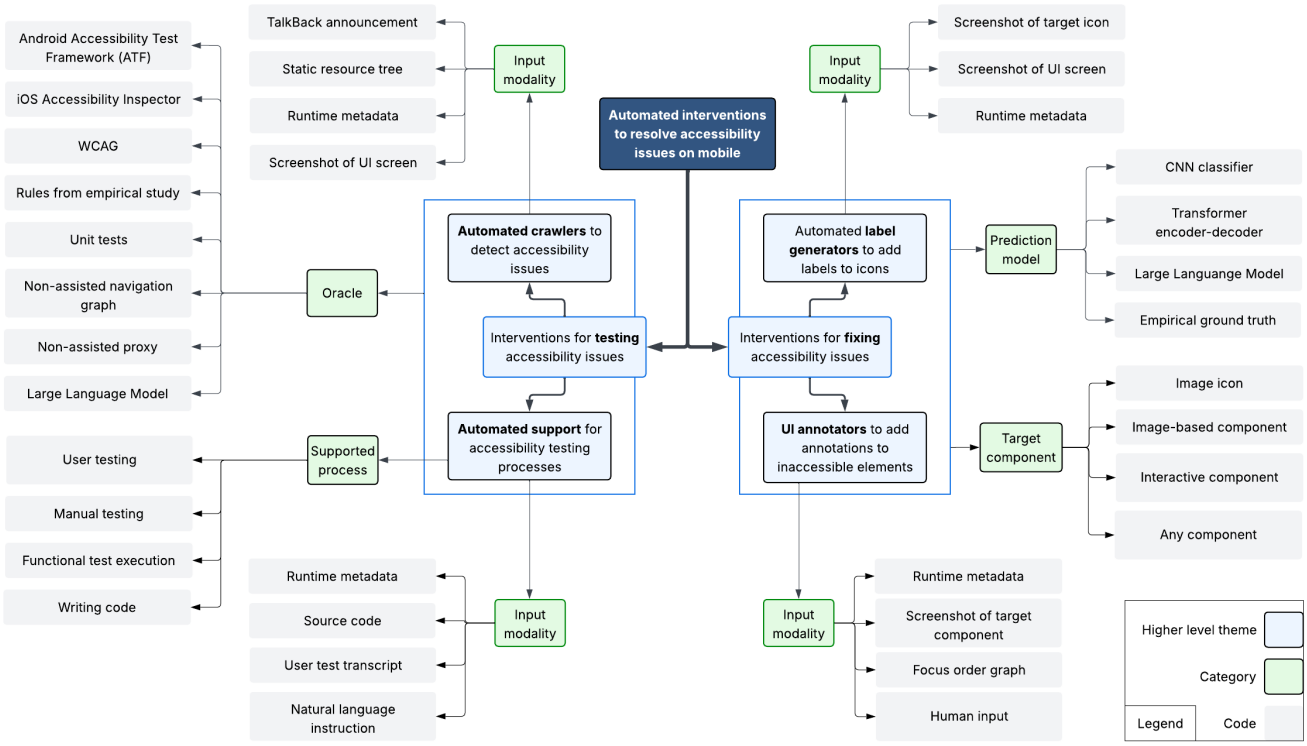


Figure 3: A map of themes, categories and codes from the thematic synthesis process for SRQ2.

To predict focusability issues in Android apps, P6 [44] developed GIFT, which constructs graph models on the focus order of the UI screen. The prediction model follows the authors' ground truth, which was constructed from user studies and a subsequent framework.

Lastly, P4 [48] developed Reca11, which employs LLM to extract perceived accessibility issues from user tests. The technique aims to streamline accessibility user testing by converting large transcripts from user test recordings into more digestible lists of issues.

A.2.3 Label Generators. P26 [61] provided a model for automatically generating natural language captions for image-based widgets like `Buttons`, `ImageButtons`, `FloatingActionButtons` and more. They employ a multimodal deep learning model with Transformer encoder-decoders and a ResNet Convolutional Neural Network (CNN), that takes in a UI screenshot of a target element and relevant properties like existing text, bounds, type of view and more.

P25 developed LabelDroid [24], which also uses Transformer encoder-decoder and a ResNet-101 CNN trained on MS COCO. However, the model is only given a screenshot of the UI element, and no contextual information.

P22's Coala [74] improved upon LabelDroid by providing contextual information to the prediction model. Along with a cropped screenshot of the target icon, the model is provided with "usage context" data comprising of app category, Activity name, screen title, element bounds, neighbouring element ids and any visible text.

P17 [25], an iOS-based approach, focused on pixel-only classification for generating icon labels. Their approach relies on the cropped-image of the target UI component, and textual and relational properties of nearby elements for icon labeling.

Lastly, P8 [65] proposed Metila, which uses a two-stream mean-teacher Transformer captioner to automatically generate labels for Android `ImageViews` and `ImageButtons`. They demonstrate better results than prior approaches despite not using contextual information.

A.2.4 UI Annotators. P30 [114] proposed the idea of interaction proxies, a runtime virtual overlay on an Android app that lets the auditor correct labels, restore missing functionality or re-order navigation. Their proof of concept was aimed at advocating for crowdsourced accessibility repair of inaccessible applications. However, the use case for the technique can also be during the testing phase of an application.

Demonstrating a similar idea to P30, P28 [115] created an annotation pipeline of UI components in runtime. Targeted towards both developers and possibly the community, the approach utilizes `AccessibilityNodeInfo` and screenshots to add or fix labels, change focus order and provide actionability to custom views.

P20 [109] presented Screen Recognition, which annotates UI elements on iOS apps based solely on the screenshot. This pixel-based system aimed to make existing inaccessible apps accessible to

VoiceOver without relying on developers. Target fixes include labeling visual elements, generating state or interactivity information, and grouping or segmenting UI elements.

To automatically detect and fix issues related to focus order, P10 [112] developed RGNF, a mechanism for redrawing GUI navigation flows. At runtime, the tool simulates Talkback swipe navigation to generate the default flow, groups navigated UI elements based on proximity and similarity of shape, and generates a new reading order according to the grouping.

HintDroid, developed by P9 [63], focused on providing intuitive hint texts to edit fields in Android. They used a retrieval-based LLM for generating the hint, by feeding it relevant contextual information from the UI runtime hierarchy.

Lastly, P5 [111] proposed another LLM-based annotator, CasGPT, for annotating the changing states of various interactive UI elements. Elements include `ButtonView`, `TextView`, `CheckBox` and `RadioButton`, and `ProgressBars`. As a prompt, the LLM is provided with the `AccessibilityNodeInfo` , type of user action, color, in-context examples and a chain-of-thought reasoning.

A.3 Accessibility Issue Types from the Systematic Literature Review

A.3.1 Labeling-related Issues. The most commonly addressed type is $I_{lb}1$, *Missing Label* — when image-based UI components do not have an alternative text. To detect these issues, testing tools look at the `contentDescription` of image-based components and check whether it is empty. To fix these issues, label generators create labels for icons using deep learning models, or UI annotators provide post-hoc annotation capabilities.

$I_{lb}2$, *Missing Hint Text* is similar to $I_{lb}1$, as it checks for missing alternative information, in this case, for edit fields. Edit fields need to be provided with a hint text that the screen reader can announce. This is tested by checking the value of the `hint` attribute in edit fields. P9 aimed to fix this issue using LLM generated text.

A similar but computationally different issue to $I_{lb}2$ is $I_{lb}3$, *Editable content description*, which checks whether the hint or label for an edit field is misplaced in the `contentDescription` attribute instead of the `hint` or `labelBy` attributes.

$I_{lb}4$, *Redundant Description* checks the quality of labels, whether the type (e.g., "Button", "EditText") or action type (e.g., "Double tap to activate") is manually inserted as the label. This is an issue since the screen readers automatically understands and announces these attributes, leading to redundant announcements.

A more subjective evaluation of the quality of labels is covered in $I_{lb}5$, *Inadequate Description*, checking whether the label correctly and completely describes the information and purpose of a UI component. For testing, P13 depended on developer inference from their generated report of Talkback announcements, while P1 relied on LLM to understand the issue. UI annotators like P28 and P30 provided pipelines for developers or third party evaluators to manually fix annotations post-hoc.

$I_{lb}6$, *Duplicate Label* checks whether there are at least two UI components on a single screen with the same alternative label, which can confuse a screen reader user in understanding the different functionality of the duplicate components.

While operating systems expose the type and action hint of native UI elements to screen readers, custom elements need to be manually exposed by developers. For $I_{lb}7$, *Unsupported Class Name*, testing tools checked the class name of the UI components and flagged any unsupported ones. In $I_{lb}8$, *Missing Interaction Types*, testing mechanisms evaluated whether a custom view's action hint could be programmatically inferred. P28 provided annotation mechanism to incorporate necessary information to such views.

A.3.2 Navigation-related Issues. The biggest issue in this category is $I_{nv}1$, *Unfocusable Interactive Element*, when an element is not reachable with screen reader swipe gestures. A common way of detecting this issue is by comparing touch navigation with screen reader navigation, either by runtime emulation (P13, P15) or graph-based analysis (P10, P19). One potential programmatic reason, reported by P15, is the inappropriate use of `importantForAccessibility` attribute, which toggles the screen reader to focus on elements. Custom views like `WebViews`, which render a web implementation into native app containers, have been reported as another common source of this issue.

Another reason for $I_{nv}1$ manifesting, but analyzed as a separate issue, is $I_{nv}2$, *Navigation Loop*. This issue occurs with manual inappropriate sequencing of UI elements using attributes such as `accessibilityTraversalBefore` or `accessibilityTraversalAfter`. This leads to the screen reader focusing only within a subset of elements when swiping right to navigate.

While these look into the lack of focusability, $I_{nv}3$ and $I_{nv}4$ look into the intuitiveness of navigation. $I_{nv}3$, *Unnatural Navigation Order* refers to directional order of focus not following intuitive logic. None of the tools in the SLR could automatically detect this issue. However, multiple annotators provide automated (P10, P20) or human-rendered (P28, P30) solution to fixing this issue.

$I_{nv}4$, *Unapparent Focus Switching*, described by P10, is the issue of focus switching between edit fields and on screen keyboard when filling up forms, disorienting the screen reader user.

Another quality assessment of navigation is $I_{nv}5$, *Excessive Interactions*, where screen reader users have to navigate through unnecessary elements to reach the target one. P15 and P21 computated the issue as 15 or more swipe gestures to reach an element. P21 further provided example cases that lead to $I_{nv}5$: when background elements are focusable during interaction with overlays, when frequently used elements like a FAB (Floating Action Button) is located at the end of the page, and grid layouts like calendar views with all the dates of a month focusable. P16 found this issue manifesting as a result of over-accessibility.

Lastly, $I_{nv}6$, *Broken Up Text*, describes the issue when a paragraph is read word by word, instead altogether. Usually caused in `WebViews`, a screen reader user needs to swipe through each word to read the whole text.

A.3.3 Activation-related Issues. $I_{ac}1$, *Ineffective Action* covers the most crucial aspect, when an interactive element, while focusable, is not interactable through screen readers. P7, P12 and P15 detected these issues by comparing touch-based interaction against screen reader-based proxies. Example reasons included `WebView` rendered interactable elements, which touch-based click runs a JavaScript

action, but is not exposed to screen readers. P30 provides a post-hoc solution to incorporate intended action as an interaction proxy layer.

Referenced from ATF, $I_{ac}7$, *Clickable Span* refers to the use of `ClickableSpans`, as a way to convert text into links. Such conversions may not render the URL accessible to screen reader users.

As part of over-accessibility, P16 detected $I_{ac}3$, *Over-Actionable* issues, where interactive elements, invisible and therefore unactionable by sighted users are exposed to screen reader users through screen reader navigation. Such issues not only disorients the user, but may also pose security risks for the apps.

A.3.4 Dynamic Change-related Issues. We followed P3 in classifying the 5 types of dynamic change-related issues, where UI elements are moved or modified without notifying the screen reader users.

$I_{dy}1$, *Latent appearing content* refers to UI elements appearing on screen after the user swiped away from its location. An example would be 'close' buttons on advertisements. This causes the user to spend more time trying to find a way out of the ad.

$I_{dy}2$, *Latent disappearing content* refers to UI elements disappearing before the screen reader user can swipe to it. An example would be a bottom menu bar that is programmed to collapse when the screen is scrolled. Programmed in consideration of sighted users' usability, this leads the screen reader user to be unable to reach the menu.

$I_{dy}3$, *Latent short-lived content* is a mix of $I_{dy}1$ and $I_{dy}2$, where elements, usually inside a parent container, is dynamically displayed and hidden. $I_{dy}4$, *Latent moving content* refers to elements that move locations when the user is navigating the page. To detect these issues, P3 analyzed snapshots of the UI to check for any changing element and checked whether the element's `accessibilityLiveRegion` attribute is properly annotated.

Lastly, $I_{dy}5$, *Latent content modification* checks whether any textual or presentation attributes of an element changes dynamically without notifying the user. This is also characterized as state change issues by P5 and P20. P5 aimed to fix this issue by annotating commonly multi-state elements with LLM. P20 also targeted such elements through pixel-based detection and annotation.

B RQ2: User Study

B.1 Known Issues in Apps Under Test

In Table 12, we list the accessibility issues we manually found or injected in the apps under test. The table contains the source/s of each issue types in the app.

B.2 User Study Protocols

Each protocol consisted of the following framework:

(1) **Introduction**, where we introduced ourselves, the scope of the study, instructed on the Think Aloud protocol [102], and asked for consent.
(2) **Scenario**, where the user was narrated a scenario, based on the app under test, to imagine before starting to use the app.
(3) **Tasks**, to guide users to inaccessible elements of the app and annotate their experience.

The four protocols contained the following scenarios for the four apps:

- **Money Manager**: *Today, we'd like to analyze the accessibility of this open-source Android application called Money Manager. With Money Manager, you can keep track of your expenses. This app has features that let you add transactions and income. You can create budget plans to keep track of your spending. The app also provides analysis of your spending habit. Now imagine you were looking for an app to keep track of your expenses for this month and you found or got recommended this app. Now start using this app as if you downloaded it on your own accord.*
- **uHabits**: *Today, we'd like to analyze the accessibility of this open-source Android application called uHabits. With uHabits, you can keep track of your habits. This app has features that let you add new habits, like waking up early or walking x amount of steps per day. You can update the habits everyday and track your progress. Now imagine you were looking for an app to keep track of your recent habits starting this month and you found or got recommended this app. Now start using this app as if you downloaded it on your own accord.*
- **AnkiDroid**: *Are you familiar with the concept of flashcards? Flashcards are small cards—either physical or digital—that contain information on both sides. One side has a question, a term, or a prompt, and the other side has the answer or an explanation. You look at the prompt on the first side (the "question") and try to recall the information. Then you flip the card to check the answer or verify if you were correct. With AnkiDroid, you can use, store and create flashcards digitally. You can also keep track of your progress on your topics of interest. Now imagine you were looking for flashcard app to start practicing on Greek mythology and you found or got recommended this app. Now start using this app as if you downloaded it on your own accord.*
- **Aard2**: *Aard2 helps you keep different dictionaries in your pocket. You can find open dictionaries and wikipedias online and download them, looking up terms and bookmarking them as you go. Now imagine you were looking for such an app to search and store articles, and you found or got recommended this app. Now start using this app as if you downloaded it on your own accord.*

Lastly, the app-specific tasks are as follows,

- **Money Manager**:
  (1) Add two expenses
  (2) Add income
  (3) Add two new Budget Categories
  (4) Check Budget Analysis
  (5) Check Monthly Expenses Analysis
  (6) Edit a Budget Category
  (7) Add a Budget Plan
  (8) Go to Settings
- **uHabits**:
  (1) Add one "Yes or No" habit
  (2) Add one "Measurable" habit
  (3) Check the added habits
  (4) Open a Habit
  (5) Edit Habit
  (6) Delete Habit

Table 12: List of all known issues in the apps under test, with associated inaccessible UI components and screens. Injected issues are italicized.

App	Issue Category	Issue Types	UI Component(s) - Screen
A_{mm}	Labeling	$I_{lb}1$ Missing Label	*Add Button - Budget Categories page, Add Button - Budget Plan page*
		$I_{lb}5$ Inadequate Description	*Add Income Button - Budget Category Info dialog*
		$I_{lb}6$ Duplicate Labels	*Product Name and Price EditText hints - Add New Transaction page*
		$I_{lb}7$ Unsupported Class Name	WebView - Monthly Expenses Analysis page
	Navigation	$I_{nv}1$ Unfocusable Interactive Element	Product Name and Price EditText labels - Add New Transaction page, Total Expenses amount text - Month's Expenses page, Charts - Budget Analysis page
		$I_{nv}3$ Unnatural Navigation Order	*Add Income Dialog*
		$I_{nv}5$ Excessive Interactions	Calendar View - Add New Transaction page
	Activation	$I_{ac}1$ Ineffective Action	*Settings Button - More Options Menu*
		$I_{ac}2$ Clickable Span	*Visit our website link - Budget Analysis page*
		$I_{ac}3$ Overly Actionable	Budget Category Info dialog
A_{uh}	Labeling	$I_{lb}1$ Missing Label	Star icon - Home page, Color picker button - Create Habit page, Habit list item - Home page, Stats icons - Habit Details page
		$I_{lb}5$ Inadequate Description	Graphics Image - Tutorial page, Colors - Color picker dialog - Create Habit page,
	Navigation	$I_{nv}1$ Unfocusable Interactive Element	Habit list headers - Home page
		$I_{nv}3$ Unnatural Navigation Order	Edit Boxes and Labels - Create Habit page
	Activation	$I_{ac}1$ Ineffective Action	*Delete button - Edit Habit page*
A_{ak}	Labeling	$I_{lb}1$ Missing Label	Add button - Home/Decks Page, Up arrow buttons - New Card Edit page
		$I_{lb}5$ Inadequate Description	*"Make field back sticky" button - New Card Edit page*
	Navigation	$I_{nv}1$ Unfocusable Interactive Element	Card page
	Activation	$I_{ac}1$ Ineffective Action	Add button - Home/Decks Page
A_{ar}	Labeling	$I_{lb}1$ Missing Label	Tab Headers - Home page
	Navigation	$I_{nv}1$ Unfocusable Interactive Element	Articles cards - Article page
	Activation	$I_{ac}1$ Ineffective Action	Bookmark button - Article page

- **AnkiDroid**:
  (1) Import an existing flashcard, namely Greek Mythology
  (2) Create two flashcards
  (3) Practice the flashcards
- **Aard2**:
  (1) Import the provided dictionary (simplewiki-20250101.slob)
  (2) Search three articles and bookmark them
  (3) Go to bookmark tab and change sorting
  (4) Delete one article from the bookmark tab

B.3 Codebook for Themes

In Table 13, we provide our codebook, consisting of the codes for four themes from inductive coding analyzing the user studies: impact, perception, suggestion and workaround. We include the definitions and examples for each code along with inclusion/exclusion criteria if applicable.

Table 13: Codes, definitions, and examples for the themes 'Impact', 'Perception', 'Suggestion' and 'Workaround'

Code	Definition	Example	Excludes/Includes
Theme - Impact			
Blocker	When an element is not accessible such that it is programmatically impossible to progress.	A button that does not register action.	*Include:* User cannot progress even with hints.
Obstruction	When an element is not accessible such that the user cannot practically progress, or cannot perceive progress.	Lack of state change or feedback to let know an action was conducted.	*Include:* When the user needs/gets hints to understand the progress and complete the task.
Unusable	When an interactive element is inaccessible such that the user cannot practically use its functionality.	A Slider element.	*Include:* Complex custom elements that require interactions (swipe gestures) not practical through screen readers.
Incorrect use	When an accessibility issue causes the user to conduct a task incorrectly.	Uses the wrong button due to labeling issue.	
Caused inefficiency	When the accessibility issue causes the user to spend more time and effort (i.e. swipes) to complete the task.	Swiped all the way up for a button, which is actually at the bottom of the page.	
Confusion	When the user explicitly mentions that they are confused on what the inaccessible element means or does.	"I don't know what this does."	*Exclude:* If their mention of confusion is related to or results in the above codes.
Bothered	When the user understands what the inaccessible element is trying to say, but mentions the inaccessibility.	"This should be labeled…" But it does not affect their task completion	
Unbothered	When the user is not bothered by the inaccessible element.	Passes through an unlabeled button even though they hear that it is unlabeled.	
Theme - Perception			
Meaningless	When the user understands what the inaccessible element is, but the announced information makes little to no sense or of no practical value.	A chart that just announces a bunch of numbers.	Unlabeled elements.
Incorrect understanding: Functionality	When the user incorrectly perceives what an element's intent and functionality is.	Thinks an unlabeled button is intended for saving a form, guessed from its location. But it performs a different functionality.	
Incorrect understanding: Information	When the user incorrectly guesses the purpose of textual information conveyed through an inaccessible element.	Thinks the announced numbers, intended to visually show progress, is gibberish when announced together.	
Unaware	When the user does not know that an inaccessible element exists, due to focusability issues.	An unfocusable text displaying total amount of expenditure.	*Exclude:* Decorative elements and elements with alternative mode of perception.
Theme - Suggestion			
Instruction desired	When the user recommends more intuitive instructions regarding an element's functionality or the requirements of a task.	No textual instruction was given to indicate a dropdown element works only after items are added in a separate window.	
Alternative interaction mode	When the user recommends an alternative method to interact with an interactive element.	Date picker should have an option for entering the numbers, as well as the provided dropdown, which is difficult to work with using a screen reader.	
Theme - Workaround			
Restart the app	Upon encountering a roadblock, the user proceeds to restart the app to see if the inaccessibility persists.		*Exclude:* Issue with Talkback not related to the app under test.
Restart device	Upon encountering a roadblock persisting despite restarting the app, the user proceeds to restart the device to see if the inaccessibility still persists.		*Exclude:* Issue with device not related to the app under test.
Customized gesture	Upon encountering a roadblock, the user performs an alternative or customized gesture to interact with the inaccessible element.	Four finger double tap when normal double tap does not work.	