UNIVERSITY OF CALIFORNIA,
IRVINE

Enhancing Manual Accessibility Testing in Mobile Apps through Synergistic Integration of Assistive Technology and Record-and-Replay Techniques

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Software Engineering

by

Ziyao He

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to express my heartfelt gratitude to my advisor, Prof. Sam Malek, for providing me with the opportunity to join his lab, even when I had little research experience. Thank you for your patience, invaluable guidance, and insightful suggestions throughout my research journey. Your willingness to share your experiences has been a constant source of inspiration.

I am grateful to Prof. Stacy Branham and Prof. Iftekhar Ahmed for their valuable feedback as members of my thesis committee. Your insights and expertise have greatly enriched my work.

I extend my sincere appreciation to my labmates Navid, Aziz, Forough, and Syed. Navid, thank you for introducing me to your project and inviting me to collaborate when I first joined the lab. Aziz, your help in setting up the lab equipment has been invaluable, and I have cherished our conversations. Forough, thank you for generously sharing your experiences and offering guidance whenever I felt lost or nervous. Syed, I have enjoyed working with you and appreciate the great times we've had exploring different activities and savoring delicious food.

To my dear friends, Jingbo, Boyang, and Xiao, I am grateful for your companionship and support. Our shared experiences have brought me so much joy, and your support has been unwavering in my research and daily life.

Finally, I would like to express my deepest gratitude to my family. I could not have embarked on this journey without your unwavering support. To my beloved parents and grandparents, although the distance between us sometimes weighs heavy on my spirit, the mere thought of you and our cherished video calls fills my heart with renewed energy and strength. Your constant encouragement has been my greatest motivation. I am grateful for the opportunity to reunite with you this summer finally.

Thank you all for your unwavering support, guidance, and love.

ABSTRACT OF THE THESIS

Enhancing Manual Accessibility Testing in Mobile Apps through Synergistic Integration of Assistive Technology and Record-and-Replay Techniques

By

Ziyao He

Master of Science in Software Engineering

University of California, Irvine, 2023

Professor Sam Malek, Chair

Accessibility is a critical software quality attribute for 15% of the world's population with disabilities. As mobile apps become increasingly important for users with disabilities, the demand for accessible mobile applications is growing. Manual accessibility testing, which involves assistive technologies, is a high-fidelity methodology that ensures the accessibility quality of shipped apps. However, implementing this approach can be arduous and demands a thorough understanding of various accessibility aspects. In addition, current automatic mobile accessibility testing tools primarily focus on identifying violations of predefined guidelines, usually overlooking how users with disabilities use mobile apps via assistive technologies. This failure to consider assistive technologies while evaluating an app's accessibility can result in missing important cues.

To address this issue, this thesis proposes a record-and-replay technique encompassing recording developers' touch gestures, replaying the equivalent actions using Android TalkBack, and generating a comprehensive visualized report incorporating different interaction modes offered by TalkBack. The evaluation of this technique on existing market apps demonstrated that the proposed technique offers valuable assistance to developers in detecting complex accessibility issues at various stages of the development process.

Chapter 1

Introduction

Mobile devices are currently the most widely used digital devices [55], with billions of people using them daily. One of the reasons for their popularity is the inclusion of touchscreens, which provide users with a rich and interactive experience. At the same time, due to the numerous capabilities of mobile devices and the complexity of user requirements, developers must test and validate their app's functionality before releasing them to target users, either manually or using automated tools. Unfortunately, developers often unintentionally overlook the evaluation of their software for people with disability, despite the fact that approximately 15% of the global population consists of individuals with disabilities [59]. Many disabled users cannot interact with mobile apps through conventional ways, e.g., touch gestures. Per law enforcement and social expectations, software engineers are obliged to design applications that are accessible to all users, irrespective of their abilities. Indeed, prior research has revealed that numerous popular apps possess accessibility issues that restrict disabled users from accessing the same digital content and functionalities as users without disabilities. [20, 49, 2].

To improve the accessibility of apps, mobile developers can refer to accessibility guidelines published by technical institutes [58], as well as tech companies such as Apple [12] and Google [7]. In addition, it is strongly recommended that developers conduct user studies with individuals, particularly those with disabilities, who utilize assistive services such as screen readers. Acknowledging the significance of human evaluation in accessibility testing, software practitioners recognize that obtaining end-user feedback can present certain challenges [15]. This challenge is particularly evident for developers in small teams with limited resources, as they may face difficulties regarding the availability and cost of finding users with diverse disabilities to participate in evaluations.

Besides user studies, mobile developers can use automatic accessibility testing tools to evaluate the accessibility quality of their apps [6, 5, 13, 14]. These tools require the developer to provide the screen or user interface that needs to be evaluated. They then check if the app violates pre-defined accessibility guidelines, such as color contrast and clickable element size. However, it's important to note that many accessibility issues may only be present when using assistive technology, such as screen readers. For example, blind users who rely on Android's TalkBack screen reader may encounter accessibility issues when navigating elements and performing actions. In some cases, TalkBack may be unable to focus on certain elements, rendering them inaccessible to blind users. Thus, solely analyzing the compliance of UI design toward accessibility guidelines may not be sufficient to reveal all the accessibility problems.

Although most automatic accessibility evaluation tools do not consider assistive technology during the evaluation process, recent studies have attempted to address this limitation [51, 1, 53]. Latte [51], for example, uses GUI (graphical user interface) test cases as input and repurposes them to execute with assistive technology for accessibility evaluation. However, its real-world applicability may be limited by the fact that a significant majority of Android developers, over 92%, do not write GUI tests [39]. Another tool, ATARI [1], assesses whether TalkBack can locate the elements of the current app screen when using Linear Navigation mode (going through elements one by one). Nevertheless, it does not consider potential actions that TalkBack users might perform, such as double-tap. Groundhog [53], on the other hand, is an app crawler that can automatically visit different screens to detect unlocatable and ineffective clickable elements when using TalkBack. One major drawback of Groundhog is that it may fail to analyze some app screens due to the use of random input generation. For example, if the current screen is the login page, Groundhog may get stuck at that screen indefinitely.

Several insights help form the idea of my thesis (1) there is a need to incorporate assistive technology when conducting accessibility evaluation for mobile apps (2) The preference for manual testing by mobile developers during the app development [31, 40, 34], (3) Mobile developers lack expertise on accessibility evaluation for their apps as prior survey reported that near 48% of Android developers attributed the lack of awareness to creating inaccessible mobile apps [2] and 45% of accessibility practitioners reported inadequate resources and experts serve as the main hinder for creating accessible apps [16].

Building on the aforementioned insights, a novel automated accessibility evaluation tool named A11yPuppetry has been implemented to aid developers in gaining insights about the created accessibility issues. A11yPuppetry employs the Record-and-Replay (RaR) technique, where the tool records user actions and replays them in a later phase. However, unlike existing RaR techniques, A11yPuppetry replays the recorded actions using assistive technology, specifically Android TalkBack, though the tool's essence can be applied to other assistive technologies as well. With A11yPuppetry, developers can assess their apps manually via touch gestures while the tool records these interactions. Later, A11yPuppetry replays the equivalent actions to the recorded touch gestures using TalkBack on a different device. More importantly, A11yPuppetry engenders visualized reports toward developers after the record and replay are accomplished, highlighting any accessibility issues that are detected. This approach enables developers to evaluate their apps' accessibility without requiring specialized knowledge or expertise and provides valuable insights for them.

Chapter 2

Background

This chapter will illustrate how blind users in Android use TalkBack to navigate through applications. TalkBack, the Android screen reader, is able to announce the textual descriptions of the elements on the current screen so users can infer the functionality of different elements based on the announcements. In addition, blind users can rely on TalkBack to perform actions on behalf of them. That said, it usually involves two stages when using TalkBack: locate the desired element and perform actions. The illustration will begin with how to locate the desired element.

2.1 Locate an element

This stage primarily serves the purpose of perceiving the content of an element and locating the desired one. When activating the TalkBack service, it will focus on the first element of the current UI hierarchy and announces its textual description. Generally, TalkBack provides four navigation modes to explore the rest of the elements.

- **LinearNavigation.** Users can go through elements one by one using the Linear Navigation Mode, which involves swiping right and left to go to the next or previous element. The determination of the previous and next elements is based on their positions in the UI hierarchy. It is important to highlight that TalkBack has the capability to automatically swipe if the next or previous elements are partially or entirely out of the screen.

- **TouchNavigation.** Users can explore the screen contents by touch. When users touch a spot on the screen, TalkBack directs its focus to the element located precisely at those coordinates. Users will employ this mode when they know the specific coordinates of the element they seek, such as the top left navigate-up button. In addition, when the linear navigation fails to focus on an element, TalkBack users might use this mode to conduct an exhaustive search for finding the desired element.

- **JumpNavigation.** Instead of going through elements one by one, users can perform a swipe up and down gesture for faster navigation. The Jump Navigation mode enables users to focus only on certain elements through exploration, such as the heading, paragraph, control, etc. Furthermore, users have the ability to adjust the granularity of announcements to enhance their understanding of the content. Instead of focusing on and announcing the entire element, users can navigate to specific levels, such as lines, words, or even individual characters within each element.

- **SearchNavigation.** Users can use keywords to search for specific elements on the screen, just like searching for matched words in the text editor. The keywords can either be typed using the keyboard or voice command, and this mode can be activated using a three-finger long-press.

2.2 Perform actions

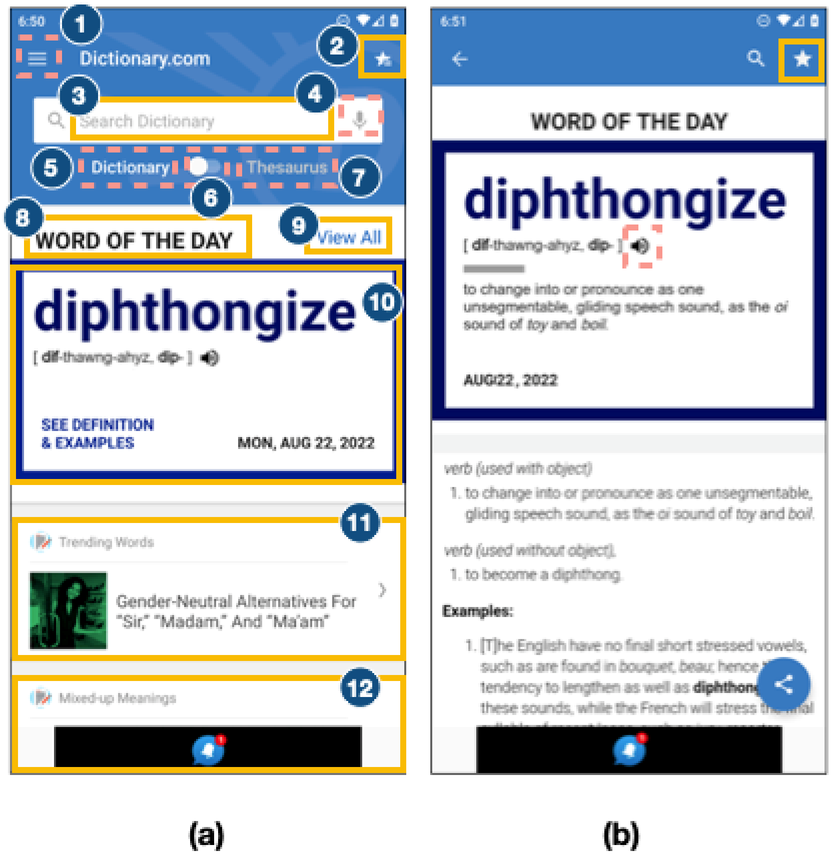
Once the desired element is located, users can rely on TalkBack to perform element-based actions on their behalf. Users can double-tap the currently focused element, and TalkBack will send a click event to the focused element. The double-tap action is the equivalent of the single-tap when using the touch gesture. Similarly, users can perform a **double-tap and hold** gesture to simulate the long press when using the touch gesture.

Besides the element-based actions, users can swipe with two fingers to replicate scrolling. In addition, TalkBack pre-defines several actions that do not depend on the app the user is using. For example, users can swipe up-and-left to return to their device's home screen.

2.3 Example

This section gives a concrete example of how to use TalkBack to navigate through a dictionary app.

Suppose blind users want to search for the word "Diphthongize" in a dictionary app. For their first use of this app, they opt for Linear Navigation mode to explore the screen elements. Initially, they listen to the textual description of the currently focused element, marked by pink box 1 in Figure 2.1 (a). TalkBack announces it as "Menu." Since the description does not match the user's expectation, they swipe right to access the next element. TalkBack then focuses on the subsequent element, denoted by box 2 in Figure 2.1 (a), and announces it as "Favorite word." Again, the description does not meet the user's expectation, so they continue swiping right to reach the next element. Upon focusing on box 3 in Figure 2.1 (a), TalkBack finally announces the element as "Search Dictionary," which aligns with the user's expectation. The desired element has been found.



to change into or pronounce as one unsegmentable, gliding speech sound, as the *oi* sound of *toy* and *boil*.

1. to change into or pronounce as one unsegmentable, gliding speech sound, as the oi sound of toy and boil.

1. to become a diphthong.

1. [T]he English have no final short stressed vowels, such as are found in bouquet, beau; hence tendency to lengthen as well as **diphthong** these sounds, while the French will stress the final

Figure 2.1: (a) The main page of Dictionary app; (b) The page after searching for the word "Diphthongize".

After locating the desired element, users can employ screen readers to perform various actions on the currently focused element. As their goal is to search for a word, users double-tap on the focused element (box 3 in Figure 2.1 (a)), which corresponds to a single-tap action for those without disabilities. Users then type "Diphthongize" in the search box and press enter on their keyboard to obtain the search result.

Then, users are directed to Figure 2.1 (b). They decide to mark "Diphthongize" as their favorite word, achievable by focusing on and double-tapping the orange box in Figure 2.1 (b). After swiping right three times to reach the orange box, TalkBack announces it as "Unlabeled Button." Consequently, blind users can not determine the functionality of the currently focused element, potentially concluding that the app does not allow them to mark words as favorites. Moreover, users wish to hear the word's pronunciation. To accomplish this, they need to locate the pink box button in Figure 2.1 (b). However, due to the developers' improper design, TalkBack only focuses on the speak button's ancestors, the blue box in Figure 2.1 (b), ignoring all descendants, including the speak button itself. This design flaw renders the functionality inaccessible for TalkBack users. Despite its critical nature, this issue goes unnoticed by Google Accessibility Scanner, the most commonly utilized accessibility evaluation tool in Android., as it does not account for assistive technology when evaluating accessibility.

The example above demonstrates that TalkBack users might require extra time and steps to achieve the same goal as a sighted user. Additionally, they face challenges when completing the same task, with some issues remaining undetectable by existing accessibility analyzers.

Chapter 3

Related Work

3.1 Accessibility Testing

Accessibility testing seeks to detect problems that prevent individuals with disabilities from effectively interacting with apps. Previous research categorized accessibility testing into two types: static accessibility testing and dynamic accessibility testing [54].

3.1.1 Static Accessibility Testing

Examining source code without executing it falls under static accessibility testing. Android Lint [9] is a static analyzer that can report accessibility issues, such as redundant labels, to developers and is incorporated into Android Studio. Previous studies also focused on certain aspects of accessibility issues. For instance, [20, 43] used deep learning techniques to detect unlabeled icons and automatically generate the associated labels for them. Nevertheless, static analysis can not detect accessibility issues only present during run-time.

3.1.2 Dynamic Accessibility Testing - Mobile

Dynamic accessibility testing involves analyzing whether an app exhibits accessibility issues by examining the attributes of rendered UI elements and their potential interactions. [5, 28, 22, 21, 33] require a single app screen as input, and then they can report accessibility issues of this screen, such as the small touch target size and low color contrast. [2, 22] employ app crawlers to explore different screens within the app and identify accessibility issues present in those screens, relieving developers from the manual assessment of each screen. Nevertheless, all the tools mentioned above failed to consider the actual interactions with the apps and ignored the role of assistive technology in assessing the accessibility of the app.

To address this shortcoming, recent studies incorporated assistive technology when evaluating the accessibility of the app. Alotaibi et al. [1] implemented ATARI to identify the unfocusable elements when using the Android screen reader TalkBack. However, ATARI requires the developer to provide the analyzed screen, and it ignores element-based actions such as clicking and typing. Latte [51] takes GUI test cases as input and repurposes them to execute with TalkBack for accessibility evaluation. However, its real-world applicability may be limited because more than 92% of Android developers do not write GUI tests [39]. Furthermore, Latte and ATARI only considered the single navigation mode in TalkBack (linear navigation) while overlooking the rest of the navigation modes. This limitation may lead to incomplete detection of accessibility issues related to different modes of interaction. Groundhog [53] is an accessibility app crawler, and it can detect unfocusable elements and ineffective actions (performed the action, but nothing happened) when using TalkBack. The drawback of Groundhog is that it can not bypass certain screens, such as the login screen, due to the random input generation technique. OverSight [42] is an automatic tool to detect overly accessible elements in Android applications. Overly accessible elements provide users of assistive technology with additional information and functions beyond what is available through conventional interaction modes. To detect OA problems, OverSight first dumps all the elements of the current window and adds potential OA elements to the lists. Then, Oversight verifies potential overly accessible elements one by one through AT on an Android emulator and generates visualized reports to offer further clues for developers. AccessiText [3] is a tool designed to detect text accessibility issues associated with the Text Scaling Assistive Service (TSAS) in Android. It executes a GUI test of an app twice, once with the default text size and another with the scaled text size. By capturing screenshots and metadata and analyzing the execution results, AccessiText determines whether text accessibility issues are present.

3.1.3 Dynamic Accessibility Testing - Web

In the realm of web applications, several accessibility testing tools have been developed to cater to the specific needs of web-based accessibility testing. MAUVE++ [19] and AChecker[24] use accessibility guidelines to assess the accessibility of websites. MAUVE++ only supports WCAG 2.1 guidelines, while AChecker enables users to specify one of nine international accessibility standards upon the accessibility evaluation and prioritize detected issues into potential, likely, and known problems. Accessibility Designer [56] enables developers to identify the discrepancies between screen reader announcements and the original text on web pages. Also, it can provide clues for developers about the ease of screen reader navigation by calculating the time required to navigate from the top to the bottom of a page. ABD [17] is a plug-in of the WebAnywhere screen reader. When screen reader users encounter accessibility issues, they can employ ABD to record the issues in the format of human-readable macros and then send the recorded issues to developers. However, ABD only supports detecting three accessibility issues, reading orders, alt text, and alt orders, respectively.

3.2 Record-and-Replay

Record-and-Replay is a technique that involves recording user actions performed during a particular session and replaying them at a later time, often on a different device or platform. Based on the way it records and replays user actions, RaR can be categorized into three types: Linux kernel, Android/Web Framework, and Application Layer.

3.2.1 Linux Kernel

Reran, Appetizer, Mosaic, and Orangutan [25, 11, 27, 35] depend on the Linux Kernel to record and replay. For instance, RERAN utilizes the ADB commands `getevent` and `sendevent` for recording and replaying events, and it must run on a rooted physical device. Since those tools are dependent on the Linux kernel, they can only capture low-level events, and it's challenging to convert low-level events to high-level events that are understandable by assistive services.

3.2.2 Android/Web Framework

VALERA [29] can record various events, including network inputs, and replay them accurately. However, the threat to the application of VALERA is the reliance on a modified Android system image. WaRR[4] relies on a customized browser as well as a browser interaction driver. WaRR uses WebKit to capture interactions and save the recorded interactions as WaRR commands. Then, the browser interaction driver will translate WaRR commands to those that browsers understand.

3.2.3 Application Layer - Mobile

Mobiplay [46] has a client-server architecture and uses screen coordinates to identify targeted elements during the record and replay stage. Espresso [8] records motion events through an attached debugger but requires the source code of the recorded app to achieve recording. Barista [23] can record and replay actions of different platforms, but due to the reliance on the Espresso framework, it fails to operate when the apps are closed-sourced. Robotium [48] can record widgets with which the users have interacted but can only capture the widgets belonging to the app's main process. In practice, a mobile application runs several processes [26]. Culebra [44] offers a desktop graphical user interface to record and replay actions and identifies widgets through the view hierarchy. However, the process for Culebra to obtain each element's view hierarchy is time-consuming and causes a large overhead. Ranorex [47] records users' interactions through the instrumentation, but the instrumentation only works if the recorded app is small.

SARA [26] employs dynamic instrumentation to record and replay multiple input sources. SARA first records the coordinates of the user interaction and then utilizes the self-replay technique to obtain the corresponding widgets under the coordinates. Then, SARA uses an adaptive replay method to replay captured events on a device with a different screen size. However, SARA's high reliance on the third-party dynamic instrumentation tool Frida makes it fail to record crucial interactions. Frida fails to instrument the class that implements the Android Interface `android.text.Editable`. RANDR [50] can record and replay various input sources, such as non-deterministic inputs, through static and dynamic instrumentation. SARA and RANDR employ instrumentation to record events and actions, so they might fail to capture events related to non-standard widgets such as `android.webkit.WebView`. Sugilite [38] employs an overlay to intercept interactions and is able to capture multiple events and elements, including the non-standard widget WebView.

3.2.4 Application Layer - Web

Several record and replay tools have been implemented to tailor for Web applications, and they operate at the application layer. Actionshot [37] and Coscripter[36] can record users' interactions on a browser and engenders human-readable scripts to replay interactions later. However, both fail to record interactions on convoluted JavaScript and HTML pages. WebVCR [10] uses the DOM signature to identify the interacted elements, requiring users to start and end each recording manually. However, certain interactions, such as HTTP authentication, may not be accurately recorded due to the reliance on DOM signatures. Smart Bookmarks [30] can automatically record users' interactions, and the recording is triggered whenever a JavaScript event happens. Smart Bookmarks uses textual description and XPath as the identifiers of the recorded elements and uses Chickenfoot's algorithm [18] to find the desired element during the replay page. WebRR [41] applies the self-replay method SARA has used and utilizes multiple identifiers to enhance the recording and replay accuracy. However, it may face challenges when recording interactions on non-standard widgets and pages, such as dynamic iframes.

Chapter 4

Overview

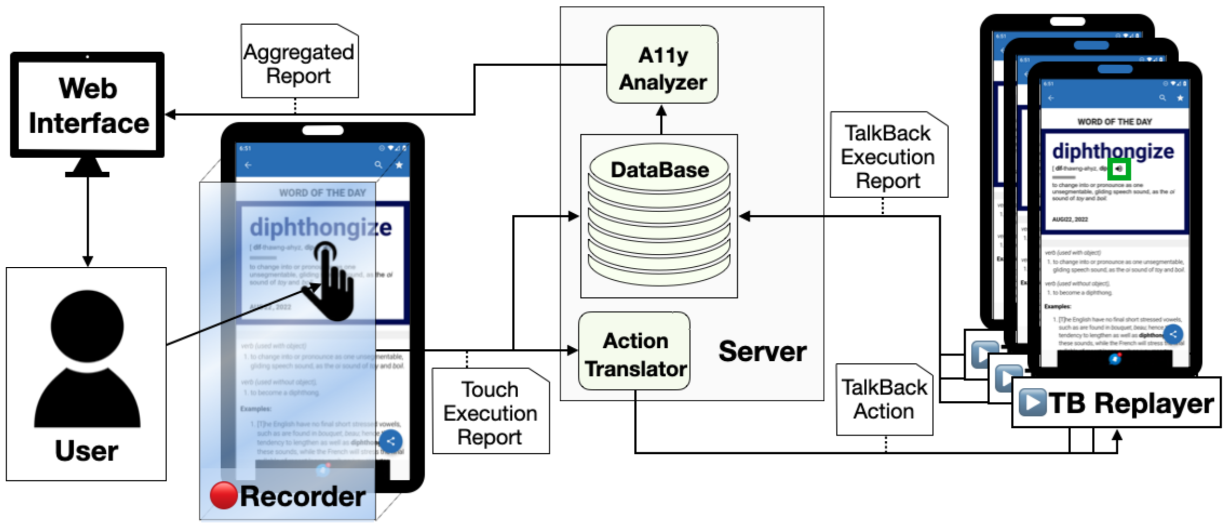


Figure 4.1: An overview of A11yPuppetry.

In this chapter, an overview of A11yPuppetry is provided. A11yPuppetry contains four crucial components, (1) Recorder, (2) Action Translator, (3) TB Replayer, and (4) Report.

To begin using A11yPuppetry, users need to enable the Recorder service. Once enabled, a transparent overlay appears on top of the app screen, intercepting the user's touch gestures. The recorder also listens for UI changes to maintain the most up-to-date state of the app.

After intercepting the touch gesture using the overlay, the recorder performs the touch gesture on behalf of the user. Then, the recorder saves the relevant information related to that touch gesture as the *touch execution report*, includes recorded actions, UI hierarchy, screenshot, and layout, and transfers the touch execution report to the server.

Upon receiving the touch execution report, the server translates the touch gesture to its equivalent *TalkBack Action*. Recall from chapter 2, TalkBack users may require extra time and steps to achieve the same goal as a sighted user. It's essential to map the touch gesture to the equivalent TalkBack Action. For example, if we touch a button, the equivalent TalkBack Action is - focuses on that button and then double-taps it.

When the translation process is completed, the server will send the TalkBack actions to replayer devices. There are four replayer devices inside A11yPuppetry, and each of them corresponds to a navigation mode in Chapter 2 (Navigation Mode refers to the way of locating the desired element). Next, every TalkBack replay service running on the replayer device performs the received TalkBack actions and transfers the TalkBack Execution Report to the server. Again, the execution report includes performed actions, UI hierarchy, screenshot, and layout.

After the recorder and replayer complete their tasks, the *A11y Analyzer* analyzes the Touch and TalkBack Execution Report and engenders a visualized report encompassing all the recording and replaying steps, along with any accessibility issues detected by A11yPuppetry. Users can access the visualized report via any web browser.

Chapter 5

Recorder

This chapter explains how the recorder captures the users' touch gestures while interacting with an app.

The recorder is built upon Sugilite [38], a PBD (Programming by Demonstration) tool that uses an overlay and Android Accessibility API to capture user interactions. To adapt Sugilite to the project's requirements, several modifications were made.

5.1 Modifications

5.1.1 Different way for performing PointGesture

The PointGesture refers to a gesture at a specific point on the screen, e.g., a single tap or long press. After intercepting the PointGesture performed by the users, the original Sugilite captures the operation and performs the operation on behalf of the users via Android Accessibility API, such as *AccessibilityNodeInfo.ACTION_CLICK*. However, if the interacted element has accessibility issues, Sugilite fails to execute the operation, causing users to remain stuck in the current window. The modified Sugilite now sends touch gestures to the coordinates of the target node instead of relying on the Android Accessibility API to perform the PointGesture. Additionally, a flag called *FLAG_NOT_TOUCHABLE* was added to the overlay while performing the gesture. This way, the touch gesture passes through the overlay to the recorded app below. Once the gesture is completed, the additional flag is removed from the overlay.

5.1.2 Precisely identify the node

The original Sugilite ignored non-clickable nodes, resulting in a *No node matched* message when users attempted to record interactions with such nodes. The modified version now considers non-clickable nodes and employs a better strategy to find the best-matched node. A node/element on the screen is added to the candidate list if the gesture coordinates are inside the bounds of that node. The nodes inside the candidate list are then sorted based on their z-index, ensuring the best-matched node is selected as the primary target.

5.1.3 Support recording global operations

Global operations are essential when navigating through an Android app. The original Sugilite does not support recording global operations, such as the global back operations. The modified Sugilite provides an option to record them and perform those global actions on behalf of users via Accessibility API, e.g., *AccessibilityService.GLOBAL_ACTION_BACK*.

5.1.4 Capture the UI snapshot of each interaction

For each interaction, the recorder captures the corresponding UI Snapshot, which consists of a screenshot and an XML layout file. The XML layout file contains all the nodes in the current window.

5.1.5 Implement Websocket

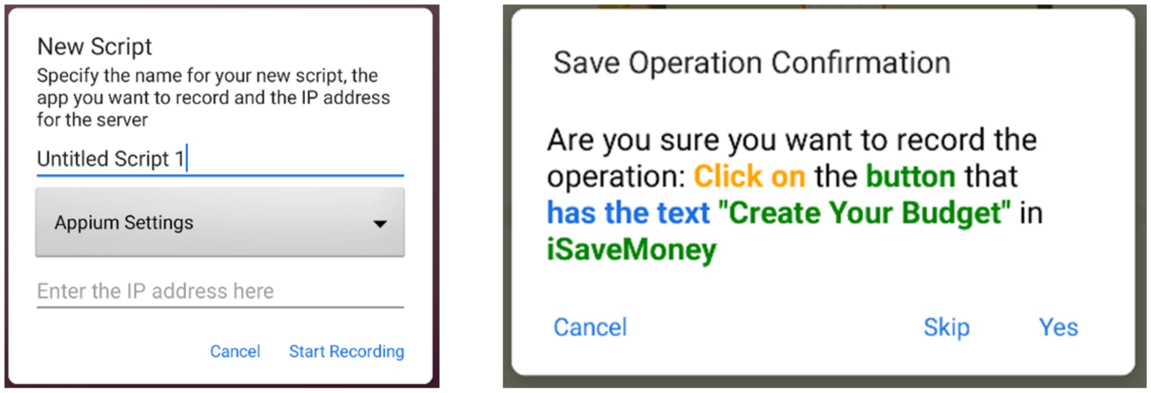
To facilitate communication with the remote server, the recorder implements WebSockets, allowing it to send the *touch execution report* to the server and receive information from it. The choice of using WebSockets protocol over HTTP is its simultaneous bi-directional communications between the client and the server. When the recorder starts, it sends the start command containing the recorded app's package name to the remote server. Also, each recorded interaction will be sent from the recorder to the remote server during the recording. After users end the recording, the end command will notify the server that the recording process is finished. The default remote server address is configured as *ws://10.0.2.2:8765/*, corresponding to the local host. Users can change the remote server address under the Settings of the recorder.

5.1.6 Support recording LineGesture

Unlike PointGesture, the LineGesture requires users to put their fingers on the screen and draw a line, e.g., Swipe-up. The recorder employs the Android *OnTouchListener* interface to intercept the LineGesture performed by the users and determines the direction based on the start and end coordinates of the drawn line. The recorder then uses the Android Accessibility API to dispatch and perform the LineGesture.

In the next section, the workflow of the recorder will be illustrated.

5.2 The recorder workflow


New Script

Specify the name for your new script, the app you want to record and the IP address for the server

Untitled Script 1

Appium Settings

Enter the IP address here

Cancel Start Recording

Save Operation Confirmation

Are you sure you want to record the operation: Click on the button that has the text "Create Your Budget" in iSaveMoney

Cancel Skip Yes

Figure 5.1: (a) The start recording interface (b) The confirmation dialog.

Users can enable the recorder service by accessing the recorder interface shown in Figure 5.1(a). In this interface, users need to specify the name of the recording, the recorded app, and the IP address of the WebSocket server. To simplify the process, the recorder provides a list of installed third-party apps on the device, eliminating the need for users to manually enter the app name. If the IP address field is left blank, the recorder will use the default remote server address configured in the recorder's settings.

Once started, the recorder runs an Android Accessibility Service in the background and draws an overlay (an Android.View Object) over the recorded app in the foreground. The registered background service of the recorder receives certain AccessibilityEvent when some noticeable events happen in the user interface. Combining the background service with an overlay, the recorder can capture user interactions in the current window. When users perform a touch gesture, the recorder intercepts that touch gesture and engenders a pop-up window to confirm whether the recorded gesture matches user expectations, as indicated in 5.1 (b). If users ensure, the recorder saves the captured gesture and performs that gesture on behalf of users. Then, the recorder sends *touch execution report*, which includes recorded actions, UI hierarchy, screenshot, and layout, to the remote server via the WebSocket.

In addition to capturing normal touch gestures, the recorder is also capable of intercepting typing interactions with the app through the AccessibilityEvent *TYPE_VIEW_TEXT_CHANGED*. This enables the recording of text inputs made by the user during the interaction with the app.

Chapter 6

Action Translator

The action translator component is responsible for translating the recorded touch gesture into an equivalent action that can be understood and performed by TalkBack. To facilitate this translation process, TalkBack actions are categorized into three types.

6.1 TalkBack Actions

- **ElementBased.** This type of action/interaction is primarily employed to comprehend the content of an element and subsequently perform specific actions on the currently focused element. As discussed in Chapter 2, TalkBack provides various effective methods for focusing on specific elements, enabling users to navigate and interact with the app's content in a more versatile manner.
  - **LinearNavigation.** Users can go through elements one by one using the Linear Navigation Mode, which involves swiping right and left to go to the next or previous element. The determination of the previous and next elements is based on their positions in the UI hierarchy. It is important to highlight that TalkBack has the

capability to automatically swipe if the next or previous elements are partially or entirely out of the screen.

  - **TouchNavigation.** Users can explore the screen contents by touch. When users touch a spot on the screen, TalkBack directs its focus to the element located precisely at those coordinates. Users will employ this mode when they know the specific coordinates of the element they seek, such as the top left navigate-up button. In addition, when the linear navigation fails to focus on an element, TalkBack users might use this mode to conduct an exhaustive search for finding the desired element.
  - **JumpNavigation.** Instead of going through elements one by one, users can perform a swipe up and down gesture for faster navigation. The Jump Navigation mode enables users to focus only on certain elements through exploration, such as the heading, paragraph, control, etc. Furthermore, users have the ability to adjust the granularity of announcements to enhance their understanding of the content. Instead of focusing on and announcing the entire element, users can navigate to specific levels such as lines, words, or even individual characters within each element.
  - **SearchNavigation.** Users can use keywords to search for specific elements on the screen, just like searching for matched words in the text editor. The keywords can either be typed using the keyboard or voice command, and this mode can be activated using a three-finger long-press.

- **TouchGestureReplication.** Users can swipe with two fingers to bypass the TalkBack overlay for replicating fling, scrolling, and dragging.
- **PredefinedActions.** TalkBack pre-defines several actions that do not depend on the app the user is using. For example, users can swipe up-and-left to return to their device's home screen.

6.2 Translation

The PointGesture and LineGesture were mentioned and explained in Section 5.1. We can translate them to the TalkBack actions that are discussed above.

6.2.1 PointGesture

Again, the PointGesture refers to the gesture that happens at a specific point on the screen. We can map PointGesture to the ElementBased TalkBack actions as PointGesture have to interact with a specific element. To precisely locate the element that has been interacted with using PointGesture, we can rely on the UI hierarchy inside the touch execution report sent from the recorder.

6.2.2 LineGesture

TouchGestureReplication and PredefinedActions can both map to Line Gesture. We can replicate the swipe gesture either through the two fingers swipe or pre-defined action, e.g., Recent Apps.

Chapter 7

Replayer

In the replay phase of A11yPuppetry, the translated TalkBack actions from the Action Translator are executed using the TB Replayer component. There are four replayer devices inside A11yPuppetry, each corresponding to a navigation mode that is explained in Chapter 2, i.e., Linear, Touch, Jump, and Search. Inside each replayer device it has TB Replayer and TalkBack. Similar to the recorder service, TB Replayer employs Android Accessibility-Service to communicate with the TalkBack and send commands to TalkBack to perform the actions received from the Action Translator.

Recall from Chapter 6, three types of TalkBack actions were introduced. To perform the TouchGestureReplication, a copy of the recorded LineGesture is made, and the coordinates of the copied LineGesture are adjusted by moving them 2cm towards the right or top of the screen. Then, we combine these two LineGestures and send a command to TalkBack to perform the TouchGestureReplication. In addition, we have a database that stores the mapping between the name of the predefined action and the corresponding actions. For example, swiping up and left enables TalkBack users to go to the device's home screen. During replay, the appropriate TalkBack action is retrieved from the database based on the recorded action's name, and the corresponding action is performed using TalkBack.

ElementBased actions pose more challenges as they require precisely locating the desired element and tracking the element's information before and after the currently focused node. Based on this, we introduced a graph model called TENG (TalkBack Element Navigation Graph). The details of TENG can be found in our recent paper [52].

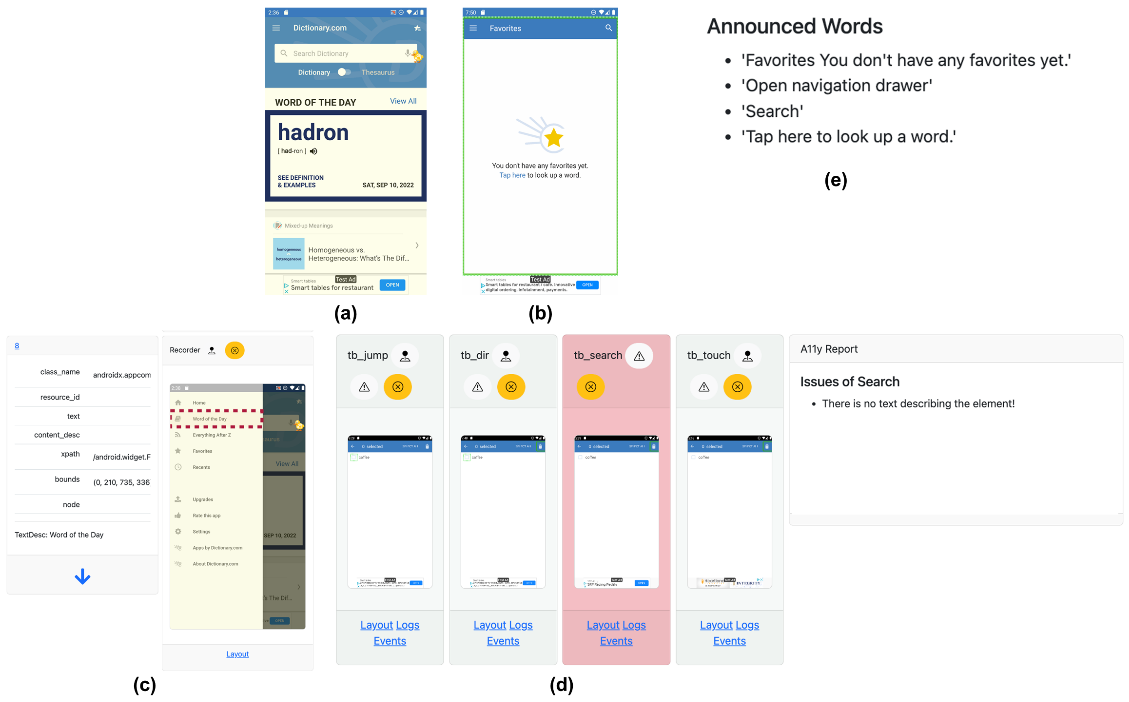
Once the replay is finished, it transfers the TalkBack Execution Report to the server, which includes performed actions, UI hierarchy, screenshot, and layout.

Chapter 8

Report

After the recorder and replayer complete their tasks, the *A11y Analyzer* analyzes the Touch and TalkBack Execution Report and engenders a visualized report encompassing all the recording and replaying steps, along with any accessibility issues detected by A11yPuppetry. Users can access the visualized report via any web browser. Since A11yPuppetry is targeted toward developers with little knowledge about accessibility and accessibility testing, we implemented the following features:

- **Annotated Video.** The A11y Analyzer generates recorded videos by utilizing the captured screenshots and annotates the interacted element if the recorded touch gesture is a PointGesture, as indicated by Figure 8.1(a). Additionally, the A11y Analyzer generates a replayer video by utilizing the captured screenshots and annotates the focused elements if the replayed TalkBack action is an ElementBaed action, as depicted in Figure 8.1(b). These annotated videos visually highlight the relevant elements and represent the interaction process.
- **Action Detail.** Besides the annotated video, the report presents action detail - the essential information of each interaction, as indicated by Figure 8.1(c). The action detail



Announced Words

- 'Favorites You don't have any favorites yet.'
- 'Open navigation drawer'
- 'Search'
- 'Tap here to look up a word.'

A11y Report

Issues of Search

- There is no text describing the element!

Figure 8.1: (a) The annotated recorder video; (b) The annotated replayer video; (c) The action detail; (d) The execution result; (e) The blindfold mode.

comprises the class name, the textual descriptions, the UI Hierarchy, and the bounds of the clicked element. The clicked element is visually highlighted with a red box, making it easy to identify.

- **Execution Result.** Since A11yPuppetry includes four TB replayer devices, each responsible for a specific navigation mode (as described in the Chapter 2), the execution result is presented separately for each replayer device. If the execution of a replayer device fails, that replayer is highlighted using the red color, as indicated by Figure 8.1(d) — the replayer uses search navigation mode failed to act. The report also includes a summary of accessibility issues detected during the execution, along with potential causes for the identified issues.
- **Blindfold Mode.** Some accessibility issues may not be evident solely by using the replayer video, such as the issues concerned with the app's semantics. When the textual descriptions of the visual icons are separate from their corresponding functionalities, it adds confusion and obstructions for TalkBack users. We implemented the blindfold that lists

the elements' textual descriptions announced by TalkBack, as indicated by Figure 8.1(e).

- **State Comparison.** A11yPuppetry compares the states of the replayers and the recorder by examining the UI hierarchy of the apps prior to each interaction. If the replayers and the recorder have different states, it may indicate potential issues. The report includes a warning sign near the replayer in case of state inconsistencies, allowing developers to investigate and address discrepancies.

Chapter 9

Evaluation

This chapter illustrates the effectiveness and the limitation of A11yPuppetry through the experiments and the user studies.

We selected five Android apps with known accessibility issues that are either detected by Groundhog [53] or reported by users on social platforms [32]. For each app, we designed a task involving the app's main functionality and incorporated the inaccessible part. The designed task for each app can be found in Appendix A. Furthermore, the first four columns in Table 9.1 summarize the information of the subject apps, which contains the app name, category, installation number, and the number of actions in the designed task.

Next, we used A11yPuppetry to record and replay the designed task for each app. The experiment devices are Pixel 4a emulators with Android version 11 and TalkBack version 12.1. We also ran the same tasks on the existing techniques Latte [51], and Google Scanner [5] to compare the performance of A11yPuppetry to them. Since Latte is a GUI test cases-oriented tool, we translated the recorded use case from the recorder to the corresponding GUI test case and fed it into Latte. For Google Scanner, we ran it on the app screens that involve the potential actions since the scanner is not a use-case-oriented tool. We filtered the reported issues from the scanner that are unrelated to blind users.

Table 9.1: The evaluation subject apps with the detected accessibility issues.

App	Category	#Installs	#Actions	#User Issues	#Scanner Issues	#Latte Issues	#A11yPuppetry Issues				
							Linear	Touch	Jump	Search	Total
ESPN	Sports	>50M	24	11	18	6	6	2	13	6	17
DoorDash	Food	>10M	23	8	22	10	9	1	13	9	15
Expedia	Travel	>10M	33	8	89	4	2	3	19	7	22
Dictionary	Books	>10M	21	8	113	6	4	2	13	5	15
iSaveMoney	Finance	>1M	21	5	35	2	10	9	10	2	11

We then conducted the user studies with TalkBack users through the Fable Platform.[1] Fable enables software companies to conduct accessibility testing and user research with disabled users. Two services of Fable were used: Compatibility Test and User Interview. For the Compatibility Test, Fable distributed our designed task and the corresponding subject app to three TalkBack users. The users performed the designed task and reported accessibility issues inside each step. After the compatibility test, we gathered all the reported issues and produced a comprehensive list of issues for each step. Finally, for each app, we scheduled a one-hour online User Interview with a TalkBack user and asked the user to share the screen during the interview. The users were asked to think aloud while performing the designed task. If the interviewee encountered an issue that hindered them from completing the rest of the tasks, we provided hints and guided them to the next step. After the TalkBack users accomplished the tasks, we asked about their specific or general experience with TalkBack and today's experience with the subject app. Each app is assessed four times: three users in compatibility tests and one in the online interview.

The reported accessibility issues by Fable users served as the benchmark. We manually went through all the reported issues and categorized them into four types: (1) **Automated Detection** - user reported issues that can be automatically detected by A11yPuppetry, (2) **Evidence Provided** - user reported issues that can not be automatically detected by A11yPuppetry but A11yPuppetry provides certain clues or evidence toward the existence of the issues in the report, (3) **Unsettled Issues** - A11yPuppetry detected such issues, but users found them not significant, and (4) **Undetected Issues** - A11yPuppetry

[1]`https://www.makeitfable.com`

could neither automatically detect nor provide any evidence. In the following sections, we explain them in detail and compare the performance of A11yPuppetry to the existing techniques.

9.1 Automated Detection

9.1.1 Missing Speakable Text

This type of issue refers to visual elements without associated content descriptions and is the most common accessibility issue among mobile apps [20]. A11yPuppetry detect missing speakable text by using the search navigation mode. If an element does not have the associated content description, it can not be searched with TalkBack.

9.1.2 Unfocusable Element

This type of issue refers to the elements on the screen that TalkBack can not focus on. Consequently, TalkBack users can not access the content nor even realize the existence of such an element. This type of issue is illustrated in Chapter 2 Figure 2.1 (b).

The severity of this issue depends on the importance and relevance of the unfocusable element to the overall app functionality. It's a severe issue for TalkBack users when the speak button in Figure 2.1 (b) is not focusable by TalkBack as blind users lose essential app functionality. By contrast, the collapse button to hide the expense details, as indicated by the red box in Figure 9.1, is a relatively trivial issue perceived by the users since it does not significantly impede users' ability to navigate and interact with the app's content.

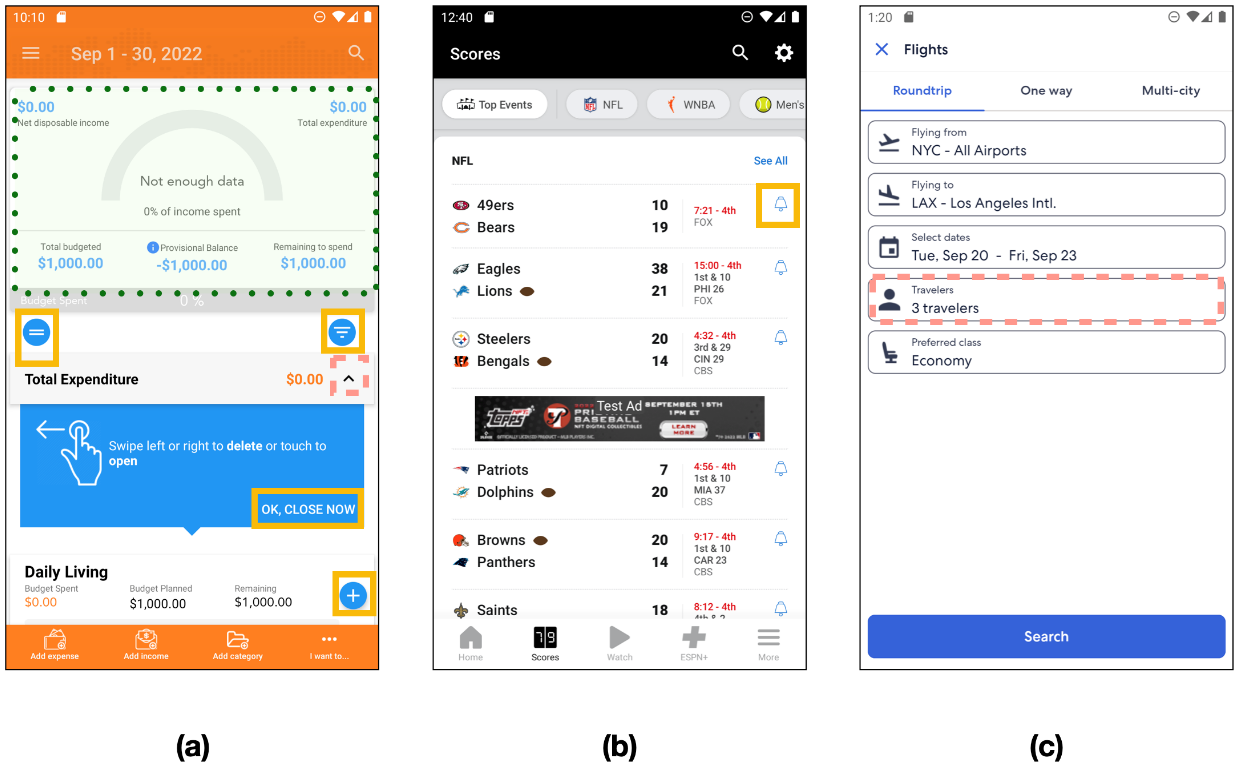


Figure 9.1: (a) The toggle button in iSaveMoney is not focusable and buttons indicated by yellow-solid boxes have ineffective actions; (b) The content description of the notification icon in ESPN has unsupported characters; (c) The textual description of travelers numbers is wrong in Expedia

9.1.3 Ineffective Action

Certain elements can be focusable by TalkBack, but nothing happens when users want to perform an action on it. For instance, users could focus on the two buttons annotated by the yellow box in Figure 9.1 (a), but nothing happened once they double-tapped them. A11yPuppetry detects this issue automatically by comparing the states of the recorder and the replayer. An ineffective action issue is detected when the states of the replayer remain the same after performing the action, but the recorder state indicates a change after performing the equivalent touch gesture.

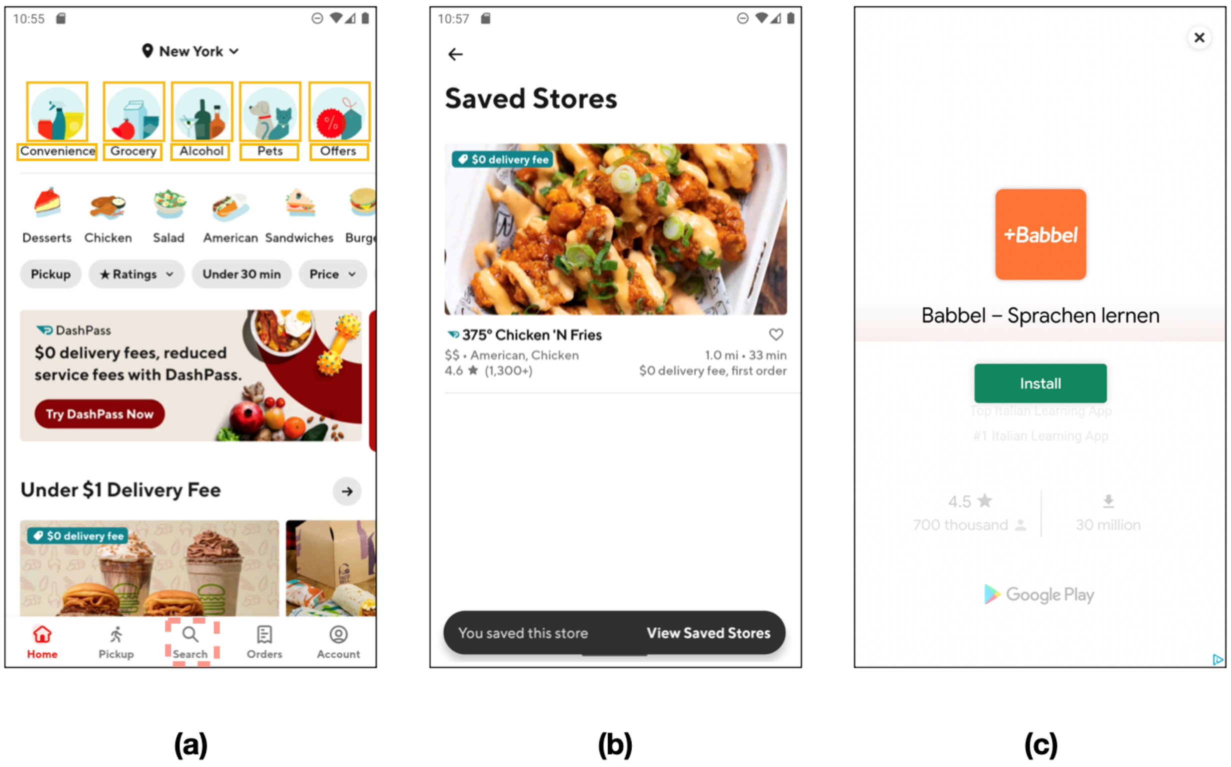


Figure 9.2: (a) After pressing the search tab in DoorDash, a new search page appears without any announcement; (b) List of saved stores in DoorDash; (c) The interstitial ad in Dictionary app and the close tab is not focusable by TalkBack.

9.2 Evidence Provided

9.2.1 Difficulties in Reading

It's essential to consider how TalkBack announces texts for blind users. Several accessibility issues were reported by users that the way TalkBack announced the text added obstructions for them to perceive the content. By analyzing the **annotated videos** and utilizing the **blindfold mode** in A11yPuppetry, developers can identify instances where TalkBack's text announcements may cause confusion or obstructions for blind users. For example, inside the Doordash app, each category (annotated by the yellow-box in Figure 9.2 (a)) is announced twice, one for the visible text, e.g., "Convenience", another for the unlabeled image, e.g., "Unlabeled". Although the app is not completely incomprehensible because of those issues, those issues add obstructions to blind users.

9.2.2 Uninformative Textual Description

Sometimes, even though a visual element has an associated label, it does not help blind users understand the content or functionality of the app better. Although A11yPuppetry can not analyze the semantics of the content description to determine if it's informative, developers can check the blindfold mode manually to determine if the textual description is informative. Following are examples of uninformative textual descriptions.

The notification icon (annotated by the yellow box) in Figure 9.2 (b) has an uninformative content description "Í". Similarly, in Figure 9.2 (c), the textual description for the travelers' information is "Number of travelers. Button. Opens dialog. 1 traveler". This description fails to accurately represent the number of travelers displayed in the figure, which is three. Blind users relying on TalkBack announcements may receive incorrect or misleading information, leading to confusion and difficulties in using the app effectively.

9.3 Unsettled Issues

A11yPuppetry has detected several jump and search navigation mode-related issues. For example, one common issue is when certain app buttons are not properly designated as buttons but instead designed as TextViews. In this way, those elements can not be located using the jump navigation mode. The users think this kind of issue is minor, as they don't use jump and search navigation modes frequently.

9.4 Undetected Issues

A11yPuppetry can not detect all the accessibility issues reported by the users, and the best way to assess the app's accessibility is by conducting studies with disabled users. Overall, there are three types of issues A11yPuppetry failed to detect.

9.4.1 Improper Change Announcement

Usually, the app's layout will constantly get changed while users are interacting with the app. Sighted users can keep track of the layout changes in apps through visual cues, but it's challenging for blind users to notice a layout change and get to know the latest state of the app. For example, when TalkBack users selected the Search Tab (annotated by the pink box in Figure 9.2 (a)), they were directed to an entirely new page without any announcement.

9.4.2 Excessive Announcement

In some scenarios, excessive announcements were made by TalkBack and overwhelmed blind users. For example, as in Figure 9.1(c), the constant announcement of "Suggestions are being loaded below" while typing the destination city name can interrupt the user's flow and make it challenging to focus on the task at hand.

9.4.3 Temporary Visible Element

Some apps generate new elements on the screens to notify of an app change and let users undo or view the change. For instance, when users save a store at DoorDash, a pop-up window appears to notify users of such changes and disappears in seconds. TalkBack users

Table 9.2: The user confirmed issues detected by Scanner, Latte, and A11yPuppetry.

App	% Intersection with User-Confirmed Issues				
	Scanner	Latte	A11yPuppetry		
			Detected	Evidence	Total
ESPN	10%	18%	18%	45%	**63%**
DoorDash	25%	25%	50%	37%	**87%**
Expedia	12%	25%	50%	12%	**62%**
Dictionary	25%	50%	50%	37%	**87%**
iSaveMoney	40%	40%	40%	20%	**60%**

were informed of this change, but the pop-up window had already disappeared when they finally navigated to the bottom of the screen.

9.5 Comparison with the Existing Techniques

We compared the performance of A11yPuppetry with Scanner and Latte. Both existing tools have their shortcomings. Since Accessibility Scanner does not consider the use of assistive technology during the accessibility evaluation, it fails to detect issues that are relevant to **unfocusable element** and **ineffective action** nor provides evidence for issues, such as **difficulties in reading**. Latte [51] assumes the availability of the GUI test cases for detecting accessibility issues in Android apps and only considers one navigation mode when locating an element. Hence, Latte can only detect unfocusable elements and ineffective actions when using linear navigation. Latte is not able to provide evidence or clues for accessibility issues **Difficulties in Reading** and **Uninformative Textual Description** that A11yPuppetry supports.

Table 9.2 reports the effectiveness of Scanner, Latte, and A11yPuppetry in detecting user-confirmed issues. The essence of the A11yPuppetry is to aid developers in gaining insights about accessibility issues inside their apps. Therefore, for A11yPuppetry, we also consider evidence of the user-confirmed issue provided if that evidence was presented in the visualized report. As a result, A11yPuppetry outperforms the existing tools in the automatic and total detection rates. A11yPuppetry detected more than 70% of issues confirmed by users on average.

Chapter 10

Discussion

This section launches discussions about the findings from the user studies that might provide insights for future researchers.

Context. The most frequent accessibility issue of mobile apps is missing speakable text [20, 2], which prevents users from locating the desired functionality and perceiving the element's content. However, in some circumstances, blind users can understand the functionality of an unlabeled element through the context. For example, the users are able to remove previously saved restaurants in DoorDash, while the removing restaurant checkbox is designed without a content description. Our interviewee still managed to locate that removing restaurant checkbox. He mentioned "That is a good layout, an accessible checkbox next to [the restaurant], which is checked unchecked. It would tell you I'm in the saved stores' section. So if I uncheck a box, it's going to remove that." This emphasizes the importance of context and how it can serve as a workaround for users to find the desired elements. Developers should still strive to provide speakable text for visual elements, but understanding the role of context can help improve accessibility.

Common Sense. During the interviews, we found interviewees sometimes locate certain elements much faster than others. For example, the navigate-up button is typically located at the top left corner of the screen. Users intuitively explored the screen and used their fingers to touch the top left corner to focus on the navigate-up button instead of relying on linear navigation. This highlights the importance of designing app layouts that align with users' common sense, making navigating and interacting with the interface easier.

Advertisement. During our experiments, we did not find any advertisements. However, we noticed an interstitial advertisement got in the way while the user searched for a word in the dictionary app during the user study, as indicated in Figure 9.2. Blind users might not realize an advertisement appears in front of them until they get stuck in the current window for several minutes. Once they realize an advertisement has been displayed to them, it's still challenging to close it and return to the previously interrupted task. One of the participants tried to focus on the ad's close button through linear and touch navigation, but TalkBack could not focus on the close button. Consequently, the participant had to close and re-open the app to continue the interrupted task.

Previous research on mobile advertisements has primarily focused on their impact on the general user group. To the best of our knowledge, only two studies have investigated ads' impact on disabled users. In one such study, Tompson and Wassmuth [57] found that a significant portion of web advertisements are presented in GIF format. What's concerning is that over half of the sampled ads in the study did not have an ALT tag assigned to them. As a result, blind users who rely on screen readers are unable to access the content of these ads. Another study used questionnaires to understand the impact of online advertisements, how screen reader users tackle web advertisements, and their preferences and suggestions regarding online advertisements [45]. Investigating the advertisement's impact on blind users is an interesting and unexplored research topic and can be researched in the future.

Chapter 11

Conclusion

This thesis proposes a record-and-replay technique called A11yPuppetry that encompasses recording developers' touch gestures, replaying the equivalent actions using Android Talk-Back, and generating a comprehensive visualized report incorporating different interaction modes offered by TalkBack. The evaluation of this technique on existing market apps demonstrated that while user studies remain the most reliable method for assessing accessibility, the proposed technique offers valuable assistance to developers in detecting complex accessibility issues at various stages of the development process.

In the future, we would like to conduct experiments with software developers and software engineering students to see to what extent A11yPuppetry is able to help them understand the accessibility issues inside their apps and help them understand the impact of their designs on the app's accessibility. In addition, we would like to extend our implementation for other assistive technologies and support other platforms.

Bibliography

[1] A. S. Alotaibi, P. T. Chiou, and W. G. Halfond. Automated detection of talkback interactive accessibility failures in android applications. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 232–243, Virtual, 2022. IEEE, IEEE.

[2] A. Alshayban, I. Ahmed, and S. Malek. Accessibility issues in android apps: state of affairs, sentiments, and ways forward. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*, pages 1323–1334, Virtual, 2020. ICSE.

[3] A. Alshayban and S. Malek. Accessitext: Automated detection of text accessibility issues in android apps. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 984–995, New York, NY, USA, 2022. Association for Computing Machinery.

[4] S. Andrica and G. Candea. Warr: A tool for high-fidelity web application record and replay. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 403–410, Hong Kong, China, 2011. IEEE.

[5] Android. Accessibility scanner - apps on google play. `https://play.google.com/store/apps/details?id=com.google.android.apps.accessibility.auditor&hl=en_US`, 2023.

[6] Android. Android accessibility overview. `https://support.google.com/accessibility/android/answer/6006564`, 2023.

[7] Android. Build more accessible apps. `https://developer.android.com/guide/topics/ui/accessibility`, 2023.

[8] Android. Espresso test recorder. `https://developer.android.com/studio/test/espresso-test-recorder`, 2023.

[9] Android. Improve your code with lint checks. `"https://developer.android.com/studio/write/lint?hl=en"`, 2023.

[10] V. Anupam, J. Freire, B. Kumar, and D. Lieuwen. Automating web navigation with the webvcr. *Computer Networks*, 33(1-6):503–517, 2000.

[11] appetizerio. Replaykit. `https://github.com/appetizerio/replaykit`, 2023.

[12] Apple. Accessibility on ios. https://developer.apple.com/accessibility/ios/, 2023.

[13] Apple. Apple accessibility. https://www.apple.com/accessibility/iphone/, 2023.

[14] Apple. Debug accessibility in ios simulator with the accessibility inspector. https://developer.apple.com/library/archive/technotes/TestingAccessibilityOfiOSApps/TestAccessibilityiniOSSimulatorwithAccessibilityInspector/TestAccessibilityiniOSSimulatorwithAccessibilityInspector.html#//apple_ref/doc/uid/TP40012619-CH4-SW1, 2023.

[15] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford. Accessibility in software practice: A practitioner's perspective. *arXiv preprint arXiv:2103.08778*, 2021.

[16] T. Bi, X. Xia, D. Lo, J. Grundy, T. Zimmermann, and D. Ford. Accessibility in software practice: A practitioner's perspective. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 31(4):1–26, 2022.

[17] J. P. Bigham, J. T. Brudvik, and B. Zhang. Accessibility by demonstration: enabling end users to guide developers to web accessibility solutions. In *Proceedings of the 12th international ACM SIGACCESS conference on Computers and accessibility*, pages 35–42, Orlando, USA, 2010. Association for Computing Machinery.

[18] M. Bolin, M. Webber, P. Rha, T. Wilson, and R. C. Miller. Automation and customization of rendered web pages. In *Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 163–172, Seattle, USA, 2005. Association for Computing Machinery.

[19] G. Broccia, M. Manca, F. Paternò, and F. Pulina. Flexible automatic support for web accessibility validation. *Proceedings of the ACM on Human-Computer Interaction*, 4(EICS):1–24, 2020.

[20] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, and G. Li. Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*, page 322–334, Virtual, 2020. ICSE.

[21] S. Chen, C. Chen, L. Fan, M. Fan, X. Zhan, and Y. Liu. Accessible or not an empirical investigation of android app accessibility. *IEEE Transactions on Software Engineering*, 48:3954–3968, 2021.

[22] M. M. Eler, J. M. Rojas, Y. Ge, and G. Fraser. Automated accessibility testing of mobile apps. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation*, pages 116–126, Västerås, Sweden, 2018. ICST.

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

[24] G. Gay and C. Q. Li. Achecker: open, interactive, customizable, web accessibility checking. In *Proceedings of the 2010 International Cross Disciplinary Conference on Web Accessibility (W4A)*, pages 1–2, Raleigh, USA, 2010. Association for Computing Machinery.

[25] L. Gomez, I. Neamtiu, T. Azim, and T. Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 72–81, San Francisco, CA, USA, 2013. IEEE, IEEE.

[26] J. Guo, S. Li, J.-G. Lou, Z. Yang, and T. Liu. Sara: self-replay augmented record and replay for android in industrial cases. In *Proceedings of the 28th acm sigsoft international symposium on software testing and analysis*, pages 90–100, Beijing, China, 2019. Association for Computing Machinery.

[27] M. Halpern, Y. Zhu, R. Peri, and V. J. Reddi. Mosaic: cross-platform user-interaction record and replay for the fragmented android ecosystem. In *2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 215–224, Philadelphia, PA, USA, 2015. IEEE, IEEE.

[28] S. Hao, B. Liu, S. Nath, W. G. Halfond, and R. Govindan. Puma: programmable ui-automation for large-scale dynamic analysis of mobile apps. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pages 204–217, Bretton Woods, New Hampshire, USA, 2014. ACM New York, NY, USA.

[29] Y. Hu, T. Azim, and I. Neamtiu. Versatile yet lightweight record-and-replay for android. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 349–366, Auckland , New Zealand, 2015. Association for Computing Machinery.

[30] D. Hupp and R. C. Miller. Smart bookmarks: automatic retroactive macro recording on the web. In *Proceedings of the 20th annual ACM symposium on User interface software and technology*, pages 81–90, Newport, USA, 2007. Association for Computing Machinery.

[31] M. E. Joorabchi, A. Mesbah, and P. Kruchten. Real challenges in mobile app development. In *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 15–24, Baltimore, MD, USA, 2013. IEEE, IEEE.

[32] K. Kaja. Doordash issue tweet. `https://twitter.com/kirankaja12/status/1551710324016836608`, 2023.

[33] KIF. Keep it functional - an ios functional testing framework. `"https://github.com/kif-framework/KIF"`, 2022.

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

[35] W. Lachance. Orangutan. `https://github.com/wlach/orangutan`, 2023.

[36] G. Leshed, E. M. Haber, T. Matthews, and T. Lau. Coscripter: automating & sharing how-to knowledge in the enterprise. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1719–1728, Florence, Italy, 2008. Association for Computing Machinery.

[37] I. Li, J. Nichols, T. Lau, C. Drews, and A. Cypher. Here's what i did: Sharing and reusing web activity with actionshot. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 723–732, Atlanta, USA, 2010. Association for Computing Machinery.

[38] T. J.-J. Li, A. Azaria, and B. A. Myers. Sugilite: creating multimodal smartphone automation by demonstration. In *Proceedings of the 2017 CHI conference on human factors in computing systems*, pages 6038–6049, Bremen , Germany, 2017. Association for Computing Machinery.

[39] J.-W. Lin, N. Salehnamadi, and S. Malek. Test automation in open-source android apps: A large-scale empirical study. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1078–1089, Virtual, Australia, 2020. ACM New York, NY, USA.

[40] M. Linares-Vásquez, C. Bernal-Cárdenas, K. Moran, and D. Poshyvanyk. How do developers test android applications? In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 613–622, Shanghai, China, 2017. IEEE, IEEE.

[41] Z. Long, G. Wu, X. Chen, W. Chen, and J. Wei. Webrr: self-replay enhanced robust record/replay for web application testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1498–1508, Virtual Event, USA, 2020. Association for Computing Machinery.

[42] F. Mehralian, N. Salehnamadi, S. F. Huq, and S. Malek. Too much accessibility is harmful! automated detection and analysis of overly accessible elements in mobile apps. In *2022 37th IEEE/ACM International Conference on Automated Software Engineering*, Rochester, Michigan, USA, 2022. IEEE, ACM New York, NY, USA.

[43] F. Mehralian, N. Salehnamadi, and S. Malek. 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*, pages 107–118, Virtual, Athens, Greece, 2021. ACM New York, NY, USA.

[44] D. T. Milano. Culebra. `https://github.com/dtmilano/AndroidViewClient/wiki/culebra`, 2023.

[45] A. S. Nengroo and K. Kuppusamy. 'advertisements or adverse-tisements?'—an accessibility barrier for persons with visual impairments. *The Computer Journal*, 62(6):855–868, 2019.

[46] Z. Qin, Y. Tang, E. Novak, and Q. Li. Mobiplay: A remote execution based record-and-replay tool for mobile applications. In *Proceedings of the 38th International Conference on Software Engineering*, pages 571–582, Texas, Austin, 2016. Association for Computing Machinery.

[47] Ranorex. ranorex. `https://www.ranorex.com/mobile-automation-testing/android-test-automation/`, 2023.

[48] RobotiumTech. robotiumrecorder. `https://github.com/RobotiumTech/robotium`, 2023.

[49] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock. An epidemiology-inspired large-scale analysis of android app accessibility. *ACM Transactions on Accessible Computing*, 13(1):1–36, 2020.

[50] O. Sahin, A. Aliyeva, H. Mathavan, A. Coskun, and M. Egele. Randr: Record and replay for android applications via targeted runtime instrumentation. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 128–138, San Diego, CA, USA, 2019. IEEE, IEEE.

[51] N. Salehnamadi, A. Alshayban, J.-W. Lin, I. Ahmed, S. Branham, and S. Malek. 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*, pages 1–11, Virtual, Okohama, Japan, 2021. ACM New York, NY, USA.

[52] N. Salehnamadi, Z. He, and S. Malek. 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*, pages 1–20, 2023.

[53] N. Salehnamadi, F. Mehralian, and S. Malek. Groundhog: An automated accessibility crawler for mobile apps. In *2022 37th IEEE/ACM International Conference on Automated Software Engineering*, Rochester, Michigan, USA, 2022. IEEE, ACM New York, NY, USA.

[54] C. Silva, M. M. Eler, and G. Fraser. 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*, pages 286–293, Thessaloniki, Greece, 2018. DSAI.

[55] StatCounter. Desktop vs mobile vs tablet vs console market share worldwide. `https://gs.statcounter.com/platform-market-share`, 2022.

[56] H. Takagi, C. Asakawa, K. Fukuda, and J. Maeda. Accessibility designer: visualizing usability for the blind. *ACM SIGACCESS accessibility and computing*, (77-78):177–184, 2003.

[57] D. Thompson and B. Wassmuth. Accessibility of online advertising: a content analysis of alternative text for banner ad images in online newspapers. *Disability Studies Quarterly*, 21(2), 2001.

[58] W3C. Web content accessibility guidelines (wcag) overview. `https://www.w3.org/WAI/standards-guidelines/wcag/`, 2023.

[59] WHO. World report on disability. `https://www.who.int/disabilities/world_report/2011/report/en/`, 2011.

Appendix A

User Study Tasks

A.1 Dictionary

1. Type the word "Coffee" in the search bar, the app should provide a list of entries, please select the first entry (which should be "Coffee").
2. Listen to the pronunciation of the word by selecting the speaker button. Then read the IPA (International Phonetic Alphabet) of the word. You may need to select the "Show IPA" link to reveal the IPA of the word.
3. On the same page, read the definition of the word "Coffee". It should start with "a beverage consisting of".
4. Mark the word coffee as a favorite word by selecting the star button.
5. Select the back or navigate up button in the app (not Android's general back button) to go to the main page. Then open the menu by selecting the navigation drawer button.
6. In the menu, select "Word of the Day". Then on the new page, select the second word in the list.
7. On this page, listen to the pronunciation of the word by selecting the speaker button. Then read the first example of this word which is located under "Examples:" section.

8. Select the back or navigate up button in the app (not Android's general back button) to go to the main page. Then open the menu by selecting the navigation drawer button. In the menu, select "Favorites".
9. On the "Favorites" page you should see the word "Coffee" which was marked as a favorite in step 4. Remove this word by selecting the edit button, then select the word "Coffee", and finally select the "Delete" button.
10. After deleting "Coffee", the favorite page should be empty with a text in the middle saying "You don't have any favorites yet. Tap here to look up a word". Please select the "Tap here" link, and search for the word "Tea".

A.2 DoorDash

1. Select the "Continue as guest" button, type "New York" in the address bar, and select the first entry (which should be "New York").
2. In the address settings page, do not change anything and select "Save". Sometimes, a pop-up window will appear to inform you, "New! Send a gift to your loved ones". In that case, select "Go Back".
3. Select the "Search" button, and type "Chicken" in the search bar. Please do not hit enter or search button once you're done typing.
4. Select the second entry of the search result. In the restaurant page, save the restaurant by selecting the save button (with a heart icon). A window appears with the title "You've saved your first store"., In this window, select "View Saved Stores".
5. Remove the saved restaurant by selecting the save button (with a heart icon). After selecting, the button should be toggled.
6. Go back to the search screen by selecting the back or "Navigate up" button two times. Then Select the "Home" button.
7. Go to the grocery category page by selecting the "Grocery" button. Then select the first

store.

8. Change the delivery option to pickup by selecting "Pickup" button (if you do not see the pickup button, try selecting another store). Once you select this button, the address of the store should be available below it under the title "This is a Pickup order"
9. Now select the info button (with an exclamation icon) under the name of the store. It should take you to a new screen with information of the store such as address and phone number.
10. Navigate back two times by selecting the "Navigate up" or back button, and finally select the "Orders" button. In the new screen, there should be a text "No recent orders".

A.3 ESPN

1. Select the "Sign Up" button, and then select the "Change" link. It may or may not ask an email, you can provide a random email just to proceed.
2. On the new window with the title "Where do you live?", choose "United States" from the drop down menu. Then select the "Done" button.
3. Then press the back button (or reopen the app) to be on the first screen. Now select the "Sign Up Later" button. It may ask you to choose a region, select any region and select next.
4. On the new screen, select one favorite league, e.g., "NBA" and then select the "Next" button.
5. On the new screen, select one team, e.g., "Lakers", and then select the "Finish" button.
6. A new screen may appear with the title "Stream your favorite teams and sports. Get ESPN+ now!". In that case, perform the back button. Now you should be on the main screen of the ESPN app.
7. Select the "Scores" button on the bottom menu, and in the "Scores" screen, select the button next to the "Top Events" button, e.g., "NFL". In the showing results, either select

the “HIGHLIGHTS” button or the notification button (with a bell icon) for one of the shown matches. Regardless of the button you selected, select the “Navigate up” or back button.

8. Select “ESPN+” on the bottom menu, select the “Settings” button, and then select the “Edition” button.
9. On the list of editions, select “Global”, a dialogue appears to ask if you want to switch, select “Continue”.
10. You should be in the main screen, select the “Search” button, and type “NFL” in the search bar. An entry “National Footbal League” should be shown under “LEAGUES” section, select that button.

A.4 Expedia

1. After opening the app for the first time, it shows an introduction page. Select the “Next” button until you reach the last screen. Then select “LET’S GO” button.
2. Close the sign-in page by selecting the “Close” button. On the main screen, select the “Flights” button.
3. On the the “Flights” page, select the “Flying from” button, type “New York” and select the first entry. Then it asks you for the destination or “Flying to”. Type “Los Angeles” and select the first entry.
4. Once you enter the airports, the app shows a calendar window to select the departure date. Select August 23rd and August 26th buttons, and then press the “Done” button.
5. Now you should be on the Flights page. Change the traveler’s number to 3 by selecting the Travelers button, then increase the number of adults by selecting the plus button two times. Then select the “Done” button.
6. Now you should be on the flight’s page. Select the “Search” button. Once the search results are provided, Then go to the main screen by selecting the “Navigate up” (or back)

button and then the “Close” button, this should close the flights page.

7. Select the “Cars” button. On the “Cars” page, select the “Pick-up” button and type “New York”. Then select the first entry (which should be “New York”).
8. On the Cars page, select the “Search” button. Then go to the main screen by selecting the “Navigate up” (or back) button and then the “Close” button.
9. Select the “Account” button. Under the “Settings” section, select “Choose a theme” button. Then select the “Dark” button and press “Done”.
10. Select the “Trips” button and then select “Sign in or create free account”.

A.5 iSaveMoney

1. Skip the tutorial by selecting Next.
2. Once you get to the actual app page of iSaveMoney, select “Create your first budget” button.
3. In the “New Budget” page, do not change the start and end dates, and just select “Next” button.
4. In the “Select Categories” page, select the “ADD” button for “Daily Living” category.
5. When the dialouge appears, type 1000 for the “Estimated Budget” field, then select the “Save” button, and finally, select the “Done” button.
6. Now, you should be on the current budget page, where the title is the current month, for example, Jul 1 - 31, 2022. Select the “Add Expense” at the bottom. If you are a screen reader user, it may be the second “Add Expense” to select.
7. In the “Add Expense” page, fill the form by picking a category (Daily Living), Writing something on the Description, e.g., “SomeExpense”, and entering 500 in the “Amounts” textbox. Finally, Select the “Save Button”.
8. Now, you should be in the budget page. Try to collapse the “Total Expendture” section, by selecting the arrow inside this section.