Accessibility of Mobile User Interfaces using Flutter and React Native

Lorenzo Perinello, Ombretta Gaggi
Department of Mathematics
University of Padua
Padua, Italy
lorenzo.perinello@studenti.unipd.it, ombretta.gaggi@unipd.it

***Abstract*—From the first static webpages to Web 3.0 and ubiquitous devices, technologies enhance and influence our lives today more than ever. Mobile devices, like smartphones, allow users to download millions of applications developed with different programming languages, frameworks, and quality standards. In this context, accessibility is a critical point that needs to be persecuted not only by developers, but even offered as an out-of-the-box feature by mobile technologies so that modern mobile applications can improve the lives of all users. In this paper, we investigate how two frameworks for cross-platform development, Flutter and React Native, address accessibility issues, and we propose some solutions when the official documentation does not.**

***Index Terms*—accessibility, cross-platform mobile frameworks, React Native, Flutter, user interface, mobile applications**

I. Introduction

The market for mobile devices is constantly increasing: the number of smartphone users is growing very fast, from more than one billion users in 2013 to more than 5 billion in 2023 [1]. In this context, accessibility, defined as the ability of a user to fully perceive, understand, navigate, and interact with digital content independently from their capabilities, is a crucial aspect for enabling all people, without exception, to enjoy the benefits of technology. Unfortunately, accessibility has not been considered from the beginning, except for small bumps on physical keyboards to facilitate orientation. The situation got even worse with touch keyboards and screens.

The first law to define rights for people with disabilities was the Americans with Disabilities Act (ADA) [2] in 1990: it federally prohibits discrimination based on disability and establishes accessibility requirements for public and private entities for the first time. Although it does not directly mention digital accessibility, it deals with widespread technologies at that time, such as telephone booths.

The first utility that included a screen reader was Microsoft Narrator, which was released in 2000; two years later, Apple introduced Universal Access in macOS devices. VoiceOver, the official Apple screen reader, was released in 2005.

In the early 2000s, despite millions of mobile devices were being sold every year [3], and the great improvements in their functionality, their accessibility struggled. Evidence of this lack of consideration for the accessibility of mobile devices can be found in the release of the first iPhone in 2007. Although this model was a historic turning point for both the mobile industry and users, it was completely inaccessible. Apple introduced VoiceOver, the first consumer screen reader to support touch screens and gestures only in the third iPhone generation in 2009. In the same way, Google introduced the TalkBack screen reader in Android in 2011.

Digital accessibility requirements were established by law with the first international standards [4] on accessibility, which mainly covered web technologies, as well as the first laws in America and Europe [5] [6]. The legislative path was consolidated in the EU with the latest European accessibility directives in 2016 and 2019: the first [7] made it mandatory for all mobile applications published by the public sector to meet the accessibility standard EN 301 54 [8], the second [9] extended the same mandatory rules to private economic operators providing essential services (transport services, electronic communication services, books and ebooks, audiovisual media services, banking services and e-commerce).

In Italy, the last monitoring of public mobile applications [10], which follows an in-depth monitoring method to verify compliance [11] with accessibility requirements, showed that 22 apps (11 iOS apps and 11 Android apps) did not respect 135 accessibility requirements. A similar situation occurs in Spain: out of 18 mobile applications in the public sector (8 Android, 10 iOS) analysed, only five reached the WCAG A level or the AA level, which are required by law [12]. Furthermore, an in-depth monitoring of 57 mobile applications (34 iOS apps, 23 Android apps) conducted in Germany found that they passed on average 81,4% of mandatory accessibility requirements [13]. Regarding non-European Union Member States, the analysis by Pedroso Carvalho et al. [14] of 10 Android applications provided by Brazil's local municipalities shows that several accessibility issues prevent users from benefitting from these apps. Another report by Balaji et al. [15] considers 20 Android apps related to e-Governance and highlights frequent accessibility unconformity on several WCAG 2.0 success criteria.

Considering the private sector, a comprehensive research by Yan et al. [16] evaluated a sample of 479 native or hybrid Android applications with IBM Mobile Accessibility Checker (MAC). The results of this study showed that about 30% of the UI components analysed have accessibility problems.

Therefore, despite regulatory and standardisation efforts, challenges still exist in ensuring complete accessibility across a vast range of mobile apps, devices, and digital services.

The paper is organized as follows: Section II discusses the related work, and Section III introduces the two analysed frameworks, Flutter and React Native, and their approach to accessibility. Section IV explains our research methodology, and Section V presents our implementation proposals. We discuss the results of our analysis in Section VI, we draw a conclusion in Section VII.

II. Related works

Several studies have been conducted to better understand the relationship between people with disabilities, particularly visual impairments, and mobile technologies. The user learning process is studied in several works: Nicolau et al. [17] analysed the typing performance of blind people and their learning process; a comprehensive analysis by Rodrigues et al. [18] investigated with a structured procedure lasting eight weeks the relationship between them and their mobile technologies and their Google Talkback learning process. Rodrigues et al. [19] presented the challenges encountered by visually impaired people when interacting with their smartphones, noting that some obstacles, e.g. the usage of some specific apps or the problem of getting lost while navigating, were also encountered by users with more experience with assistive technologies. Instead, both studies are based on a small sample and require further investigation.

Jain et al. [20] reported specific mobile device usage patterns and a few related needs common among users with visual impairments who use Android TalkBack. The analysis of voice input on mobile devices proposed by Shiri et al. [21] shows that this input mode is widespread and generally satisfactory among visually impaired users. However, some features, such as editing or text verification, require more effort from users.

An in-depth analysis of the performing of motion gestures by users with visual impairment with their mobile phones is carried out by Nem Khan Dim et al. [22] who concluded that their use is more efficient than classical graphical interfaces.

Preferences for different wireless technologies, such as smartphones, of consumers with audio or vision impairment are reported by J. Morris et al. [23] which shows that iOS is the preferred mobile operating system by these users. An overview of assistive technologies, which includes an analysis of the state-of-the-art assistive tools for mobile technology, was proposed by Csapó et al. [24].

Accessibility is studied in [25], a survey of touchscreen technologies widely used in mobile or wearable devices which examined accessibility solutions and issues of various interaction scenarios involving input of data and output information.

Another review of smartphone-based assistive solutions focused on people with visual impairment analysed the current state-of-the-art of assistive devices and found out that safe travel in an unfamiliar environment (independent life is a human right recognised in the Convention on the Rights of Persons with Disabilities [26]) is still an open challenge [27].

Accessible user interfaces are discussed in several works: Mi et al. [28], proposed a checklist for the development of graphical interfaces. Image-based buttons are analysed on 5,753 Android apps with an epidemiology-like framework: missing, duplicate or informative labels are searched on the source code, finding out that 45.9% of analysed apps had less than 10% of assessed image-based buttons labelled [29]. Another more complex work is proposed by Wang et al. [30] which scrapes 19,127 apps on Play Store and, by analysing screenshots of 10408 of them with a Convolutional Neural Network and several other techniques, find out that 77.38% of them have image buttons or clickable images without any semantic label. [31] involved 41 blind participants and found out that several usability and user interface facets, such as clarity of commands and labels or an easy-to-learn and interpret workflow, have positive effects on users' satisfaction with the usage of mobile touchscreen interfaces.

Several research studies [32] [33] [34] focused on the analysis of the accessibility guidelines from W3C, their impact and possible improvement. Díaz-Bossini et al. [35] analysed guidelines by highlighting crucial aspects for older adults and developed age-centred design guidelines helpful in testing the accessibility of mobile applications.

The aspect of accessibility evaluation is also taken into consideration by many research papers: Silva et al. [36] evaluated the effectiveness of several testing software and their coverage of the BBC Mobile Accessibility Guidelines [37] and WCAG 2.0 standards. In other cases, new tools are proposed, such as MATE, presented by Eler et al. [38], which can automatically explore an Android application and perform automatic accessibility tests on the UI Widgets analysed.

[39] investigates another exciting aspect: by studying 96 threads of a mailing list and with 18 semi-structured interviews, the study finds that programmers with visual impairments have several impacts on their works and experience because accessibility features are only partially supported.

React Native and Flutter, two frameworks for mobile applications, are studied and compared in few works. Kishore et al. [40] study performance, battery consumption, CPU and storage requirements by analysing two developed apps: they found out that React Native requires more computational effort, but Flutter requires more memory space. The automatic testing perspective of React Native and Flutter is studied by Zahra et al. [41]: the paper highlights that there are very few automatic tools to test accessibility of mobile applications and that React Native outperformed Flutter regarding reusability and compatibility of automatic tests.

In this paper we will analyse how React Native and Flutter allow to create accessible user interfaces and if they provide specific documentation, tutorials or examples to help the programmers. If not, we will try to fill this gap.

III. Frameworks for cross-platform development: Flutter and React Native

A. Flutter

Google released Flutter[1], an open-source software development kit (SDK,) in 2018. It allows developers to create

[1]https://flutter.dev/

cross-platform applications for mobile, web, and desktop environment from a single codebase written in Dart. Flutter creates widgets as basic logical-graphic units to develop user interfaces: a single widget describes the aesthetic properties that will be displayed to the user based on its configuration and any changes in its state. Flutter defines two types of widgets:

- *stateless*: widgets which do not change over time and whose graphical properties depend only on their initial configuration (e.g., a simple icon or a paragraph of static text that does not change over time);
- *stateful*: widgets that change dynamically according to their state, which may change due to user interactions, data fetching from external API or sensors, etc.

The entire composition of the user interface can be visualized as a widget tree, i.e. a tree data structure in which each node represents a widget and edges between nodes define a composition relationships between them.

Flutter deals with accessibility by automatically creating an accessibility tree starting from the widget tree: each widget represents a semantic node, which implicitly exposes some information to assistive technologies by default. Moreover, this information can be enhanced, removed or modified by encapsulating UI widgets into other widgets specifically designed to improve accessibility:

- *Semantics*: annotates the semantic node with several accessibility properties according to a specified configuration defined by the developer; this configuration is defined by the SemanticsProperties class which allows to set several semantic properties and event handlers also for customising assistive technologies behaviour;
- *MergeSemantics*: merges the semantics properties of a node's semantic children;
- *ExcludeSemantics*: remove semantic information from a node's semantic children;
- *BlockSemantics*: remove semantic information from a node semantic parents which are in the same container;
- *IndexedSemantics*: add indexing information to all semantic children of a node.

B. React Native

In 2015, two years after the release of React library for web applications, Facebook released React Native[2], an open-source framework for developing mobile applications. Heavily inspired by React, React Native combines a JavaScript codebase, a native look and feel and great performances derived from its cross-compiled approach. The creators of React Native describe its main features as declarative, component-based, quick to develop and portable. Nowadays, this framework's popularity is high in the developers' community, with over 100,000 stars on GitHub's official repository and 20,000 forks.

The React Native architecture contains 3 different layers:

- *TSX/JSX Layer* this layer contains the UI components written by developers in TSX or JSX;
- *JavaScript Interface (JSI)*: this layer allows the communication between Kavascript and C++ code. It computes two steps:
  1) *Render*: the React Element Tree is created in plain Javascript (a tree of React Component as a node) and, from this, a React Shadow Tree is created by the Fabrique rendering system; the latter tree is written in C++ and is composed of Shadow Node that corresponds to React Component.
  2) *Commit*: this steps is composed of two operations: layout calculation and tree promotion. The first calculates each React Shadow Node's position, size and constraint; in the second, React Shadow Tree is promoted as the next to be mounted. This means that it has all the information to be mounted and represents the latest logical state of the React Element Tree.
- *Mount*: in this layer, the React Shadow Tree is transformed into a Host View Tree with rendered pixels on the screen. The Host View Tree will be composed of native UI components for each operative system.

In React Native, user interfaces are designed using components; each component describes the aesthetics and behaviour of a portion of the application. Components can also have a state, i.e. persistent information, and properties, i.e., data that are received as input from other components. A graphical interface is thus an aggregation of components, which in turn may be made of several components that are bound together by logical parent-child composition relationships.

The approach for managing accessibility aspects in React Native follows the concept of directly adding properties to a component to enhance its accessibility: 38 properties can be added to a component, 8 are dedicated only to iOS, 7 to Android, and 23 are for both operative systems. Each of these properties can be applied directly to every React Native component without adding or encapsulating other components to the user interface. React Native does not provide an accessibility tree or any other hierarchical data structure helpful in understanding the semantics of an app.

IV. Method

The aim of this research is to analyse accessibility aspects of different widgets and components of Flutter and React Native. The goal is to answer the following research questions:

- *RQ1*: Are the widgets and components provided directly by the frameworks accessible by default?
- *RQ2*: If a widget or a component is not accessible by default, is it possible to make it accessible?
- *RQ3*: If a widget or a component is not accessible by default, and can be made accessible by developers, how much does it cost - in terms of additional required code?

To answer these questions, we analysed a small set of components and widgets, which provides an implementation of textual contents with different purposes and semantic meanings. In particular, we analysed:

[2]https://reactnative.dev/

- *Heading*: heading elements are used to specify the title of an application section or region; in HTML they are implemented with tags `<h1>`, . . ., `<h6>`;
- *Language declaration*: different text elements can use different languages hat must be declared. In HTML, the `lang` attribute is used to define the language of a tag.
- *Text abbreviation*: Abbreviations and acronyms must be declared; `title` attribute and `<abbr>` tag are used in HTML to represent this information.

The accessibility guidelines and success criteria (WCAG) that these elements must follow to be considered accessible are listed in Table I. In the following experiment, we analysed widgets and components provided by Flutter and React Native for implementing headings, language declarations, and abbreviations to check if they are compliant with the guidelines. All the information is taken from the official websites and documentation. When a widget or a component is not fully compliant with WCAG criteria, we provide an out-of-the-box, fully-accessible implementation.

Accessibility was manually tested through the use of assistive technologies. We performed these tests in two different devices: Apple iPhone XR, with iOS 16, provided with the embedded VoiceOver screen reader and a Huaweri P8 lite 2017, Android 7.0, supplied with Google Talkback service. We initially checked if they express by default their actual semantic meaning and expose it to users and assistive technologies. If not, we developed a brief and fully accessible implementation of the same component or widget.

Since we are interested in evaluating if the two frameworks allow the development accessible apps, we do not consider third-party modules, libraries or packages. For the same reason, we do not take into account issues regarding colour, size, and other aesthetical properties since their compliance with WCAG strictly depends on designer's choices and not on the framework's features. Finally, we measured the number of code lines that must be inserted into the accessible implementation to investigate their potential dissimilarity in terms of verbosity.

V. New solutions to implement a small set of WCAG with two frameworks

A. Heading

1) Flutter: Flutter does not provide a widget designed to define headings. The solution we propose to achieve accessibility is to wrap a `Text` widget into a `Semantic` widget with the header property set to true (see Listing 1).

```
Widget build(BuildContext context) {
    return
    Semantics(
        header: true,
        child: const Text('Title')
    );
}
```

Listing 1. Definition of a heading using Flutter

TABLE I
WCAG 2.2 criteria

Element	WCAG criteria
Heading	1.3.1 2.4.6 2.4.10
Text language	§1.4 §3.1
Text abbreviation	§1.4 §3.1

2) React Native: Neither React Native provides a component for representing a heading, but we can achieve the same semantics by adding to a `Text` component a property `role` or `accessibilityRole` set to `heading` (see Listing 2).

```
return (
<Text role="heading">
This is the first heading </Text>
)
```

Listing 2. Definition of a heading using React Native

B. Text language

Both the frameworks use a very similar approach for declaring the use of a different language in a piece of text. Flutter requires to wrap the `Text` widget with a `Semantic` widget with a `attributedLabel`. The definition is rather complex and can be seen in Listing 3.

```
Widget build(BuildContext context) { return Semantics(
    attributedLabel: AttributedString(
    "Web Content Accessibility Guidelines", attributes: [
    LocaleStringAttribute(
    range: const TextRange(start: 0, end: 35), locale:
     const Locale("en"))
    ]),
    excludeSemantics: true,
    child: const Text('Web Content Accessibility Guidelines
     '));
}
```

Listing 3. Declaration of the used language in Flutter

The same result can be more easily achieved in React Native by adding to a `Text` component the `accessibilityLanguage` property with a string that contains the desired language (see Listing 4). Since this solution is very similar to the one provided by HTML language, we classified this component as accessible by default.

```
return <Text accessibilityLanguage="fr-FR">Ceci est un
    texte en francais</Text>
```

Listing 4. Declaration of the used language in React Native

C. Text abbreviation

In Flutter, an accessible abbreviation or acronym can be implemented with a `Text` widget enhanced with the `semanticsLabel` attribute which explains the meaning of the abbreviation or acronym. In React Native, this data is inserted in the attribute `aria-label` or `accessibilityLabel` of a `Text` component.

VI. Discussion

If we analyse the solution proposed in the previous section, we find there is only one situation in which components or widgets are accessible by default and expose the right semantics, i.e., the use of React Native to declare the language with a simple attribute like HTML does. In all the other situations, complete accessibility can be reached only by adding supplemental code, as shown in Table II.

TABLE II
ACCESSIBILITY OF COMPONENTS AND WIDGETS

	React Native		**Flutter**	
	RQ1	*RQ2*	*RQ1*	*RQ2*
Heading	✗	+1P	✗	+1W +1P
Text language	✔		✗	+1W +1P
Text abbreviation	✗	+1P	✗	+1C +1P

Legend: ✔: accessible by default, ✗: not accessible,
P: property, *W*: widget.

Flutter and React Native have two different approaches on accessibility governance, and this had a direct and significant impact on the experiment's results:

- *Flutter* has a "wrapped" approach in which widgets `Semantics` plays a crucial role in adding semantics and thus accessibility to widgets that do not have that;
- *React Native*, like HTML, prefers an "enhancing" approach in which a large number of properties can be added without the need of additional components.

The first approach certainly has an initial disadvantage: wrapping a widget with a `Semantics` widget increases the verbosity of the code and decreases its clarity and readability without offering any advantage if not the explicit creation of a semantic node. However, the second approach is more concise and easy and, thus, creates less verbose code. Flutter approach side effects in terms of additional lines of code compared to React Native is straightforward as shown in Table III.

Considering that the lack of resources (mainly time and knowledge about frameworks and languages) is listed as one of the main barriers for developers, a more efficient approach to accessibility is certainly a factor that can make the development of fully accessible mobile applications easier. Moreover, the benefits of a less verbose code are proven both from the point of view of understandability [42], developers mental effort [43] and software quality [44].

TABLE III
LINES OF CODE (LOC)

	React Native	**Flutter**	Δ (LOC)
	RQ3	*RQ3*	
Heading	7	11	3
Text language	7	21	14
Text abbreviation	7	14	7

VII. Conclusion

In this paper, we study if the cross-platform development frameworks Flutter and React Native allow to achieve accessibility of user interface of mobile applications. Even if we only consider a limited set of situations, our analysis shows that both frameworks do not provide a complete accessibility by default (RQ1), but we were able to find solutions to make components and widgets accessible (RQ2). Moreover, our analysis shows that React Native offers a more concise approach which leads to a briefer and thus readable source code (RQ3).

In conclusion, accessibility of mobile apps is not only a compliance requirement, but also an ethically wise decision that enhances users' experience. By ensuring that mobile applications are accessible to everyone, developers can reach a more extensive user base, provide equal opportunities for everyone, and simultaneously attain new business chances. The advancements in technology and the growing awareness of accessibility, both from the legislative and technological point of view, offer a promising path towards a more inclusive digital landscape. In this context, Flutter and React Native fit in as new, modern and popular frameworks for developing mobile applications. Our future work will study accessibility of other kind of components and widgets and provide solutions when not natively available.

References

[1] J. Degenhard, "Number of smartphone users worldwide 2013-2028," June 2023. [Online]. Available: https://www.statista.com/forecasts/1143723/smartphone-users-in-the-world

[2] "Americans with disabilities act of 1990," 42 U.S.C. § 12101, 1991.

[3] Nokia, "Nokia annual report 2000," January 2001. [Online]. Available: https://web.lib.aalto.fi/fi/old/yrityspalvelin/pdf/2000/Enokia.pdf

[4] W. W. W. C. W. A. I. Group, "Web content accessibility guidelines (wcag) 2.2," July 2023. [Online]. Available: https://www.w3.org/TR/WCAG22/

[5] U. Congress, "Section 508 of the rehabilitation act," 1998. [Online]. Available: https://www.law.cornell.edu/uscode/text/29/794d

[6] I. Parliament, "Legge 9 gennaio 2004, n. 4," January 2004. [Online]. Available: https://www.gazzettaufficiale.it/atto/serie_generale/caricaDettaglioAtto/originario?atto.dataPubblicazioneGazzetta=2004-01-17&atto.codiceRedazionale=004G0015&elenco30giorni=false

[7] European Parliament, "Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies," October 2016. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016L2102

[8] European Commission, "European standard EN 301 549 V3.2.1, Accessibility requirements for ICT products and services," March 2021. [Online]. Available: https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf

[9] E. Parliament, "Directive (eu) 2019/882 of the european parliament and of the council of 17 april 2019 on the accessibility requirements for products and services," April 2019. [Online]. Available: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32019L0882

[10] AGID - Agenzia per l'Italia Digitale, "Relazione alla Commissione Europea articolo 8, paragrafo 4, della direttiva (UE) 2016/2102," 2022. [Online]. Available: https://www.agid.gov.it/it/design-servizi/accessibilita/monitoraggio

[11] European Parliament, "Commission Implementing Decision (EU) 2018/1524 establishing a monitoring methodology and the arrangements for reporting by Member States in accordance with Directive (EU) 2016/2102 of the European Parliament and of the Council on the accessibility of the websites and mobile applications of public sector bodies," October 2018. [Online]. Available: http://data.europa.eu/eli/dec_impl/2018/1524/oj

[12] Observatorio de Accesibilidad, "Report on the result monitoring Period 2020-2021," 2021. [Online]. Available: https://administracionelectronica.gob.es/pae_Home/pae_Estrategias/pae_Accesibilidad/Informe-Resultado-Seguimiento/Resultados-Seguimiento.html?comentarioContenido=0

[13] Bundesministerium für Arbeit und Soziales, "Report of the Federal Republic of Germany to the European Commission about the periodic monitoring of compliance with the accessibility requirements of Websites and mobile applications of public bodies pursuant to Article 8 of 2021 Directive (EU) 2016/2102," December 2021. [Online]. Available: https://www.bfit-bund.de/DE/Downloads/eu-bericht-pdf.pdf?__blob=publicationFile&v=2

[14] L. P. Carvalho, B. P. M. Peruzza, F. Santos, L. P. Ferreira, and A. P. Freire, "Accessible smart cities? inspecting the accessibility of brazilian municipalities' mobile applications," in *Proceedings of the 15th Brazilian Symposium on Human Factors in Computing Systems*, ser. IHC '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/3033701.3033718

[15] V. Balaji and K. Kuppusamy, "Accessibility analysis of e-governance oriented mobile applications," in *2016 International Conference on Accessibility to Digital World (ICADW)*, 2016, pp. 141–144.

[16] S. Yan and P. G. Ramachandran, "The current status of accessibility in mobile apps," *ACM Trans. Access. Comput.*, vol. 12, no. 1, feb 2019. [Online]. Available: https://doi.org/10.1145/3300176

[17] H. Nicolau, K. Montague, T. Guerreiro, A. Rodrigues, and V. L. Hanson, "Typing performance of blind users: An analysis of touch behaviors, learning effect, and in-situ usage," in *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, ser. ASSETS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 273–280. [Online]. Available: https://doi.org/10.1145/2700648.2809861

[18] A. Rodrigues, K. Montague, H. Nicolau, and T. Guerreiro, "Getting smartphones to talkback: Understanding the smartphone adoption process of blind users," in *Proceedings of the 17th International ACM SIGACCESS Conference on Computers & Accessibility*, ser. ASSETS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 23–32. [Online]. Available: https://doi.org/10.1145/2700648.2809842

[19] A. Rodrigues, H. Nicolau, K. Montague, J. Guerreiro, and T. Guerreiro, "Open challenges of blind people using smartphones," 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1909.09078

[20] M. Jain, N. Diwakar, and M. Swaminathan, "Smartphone usage by expert blind users," in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, ser. CHI '21. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3411764.3445074

[21] S. Azenkot and N. B. Lee, "Exploring the use of speech input by blind people on mobile devices," in *Proceedings of the 15th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2513383.2513440

[22] N. K. Dim and X. Ren, "Designing motion gesture interfaces in mobile phones for blind people," *Journal of Computer Science and Technology*, vol. 29, no. 5, pp. 812–824, Sep 2014. [Online]. Available: https://doi.org/10.1007/s11390-014-1470-5

[23] J. Morris and J. Mueller, "Blind and deaf consumer preferences for android and ios smartphones," in *Inclusive Designing*, P. M. Langdon, J. Lazar, A. Heylighen, and H. Dong, Eds. Cham: Springer International Publishing, 2014, pp. 69–79.

[24] Á. Csapó, G. Wersényi, H. Nagy, and T. Stockman, "A survey of assistive technologies and applications for blind users on mobile platforms: a review and foundation for research," *Journal on Multimodal User Interfaces*, vol. 9, no. 4, pp. 275–286, Dec 2015. [Online]. Available: https://doi.org/10.1007/s12193-015-0182-7

[25] W. Grussenmeyer and E. Folmer, "Accessible touchscreen technology for people with visual impairments: A survey," *ACM Trans. Access. Comput.*, vol. 9, no. 2, jan 2017. [Online]. Available: https://doi.org/10.1145/3022701

[26] United Nations, "Convention on the rights of persons with disabilities," Dec. 2006.

[27] A. Khan and S. Khusro, "An insight into smartphone-based assistive solutions for visually impaired and blind people: issues, challenges and opportunities," *Universal Access in the Information Society*, vol. 20, no. 2, pp. 265–298, Jun 2021. [Online]. Available: https://doi.org/10.1007/s10209-020-00733-8

[28] N. Mi, L. A. Cavuoto, K. Benson, T. Smith-Jackson, and M. A. Nussbaum, "A heuristic checklist for an accessible smartphone interface design," *Universal Access in the Information Society*, vol. 13, no. 4, pp. 351–365, Nov 2014. [Online]. Available: https://doi.org/10.1007/s10209-013-0321-4

[29] A. S. Ross, X. Zhang, J. Fogarty, and J. O. Wobbrock, "Examining image-based button labeling for accessibility in android apps through large-scale analysis," ser. ASSETS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 119–130. [Online]. Available: https://doi.org/10.1145/3234695.3236364

[30] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind your apps: Predicting natural-language labels for mobile gui components by deep learning," ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 322–334. [Online]. Available: https://doi.org/10.1145/3377811.3380327

[31] A. Khan and S. Khusro, "Blind-friendly user interfaces – a pilot study on improving the accessibility of touchscreen interfaces," *Multimedia Tools and Applications*, vol. 78, no. 13, pp. 17 495–17 519, Jul 2019. [Online]. Available: https://doi.org/10.1007/s11042-018-7094-y

[32] M. Ballantyne, A. Jha, A. Jacobsen, J. S. Hawker, and Y. N. El-Glaly, "Study of accessibility guidelines of mobile applications," in *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia*, ser. MUM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 305–315. [Online]. Available: https://doi.org/10.1145/3282894.3282921

[33] N. Alajarmeh, "The extent of mobile accessibility coverage in wcag 2.1: sufficiency of success criteria and appropriateness of relevant conformance levels pertaining to accessibility problems encountered by users who are visually impaired," *Universal Access in the Information Society*, vol. 21, no. 2, pp. 507–532, Jun 2022. [Online]. Available: https://doi.org/10.1007/s10209-020-00785-w

[34] B. Grellmann, T. Neate, A. Roper, S. Wilson, and J. Marshall, "Investigating mobile accessibility guidance for people with aphasia," in *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, ser. ASSETS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 410–413. [Online]. Available: https://doi.org/10.1145/3234695.3241011

[35] J.-M. Díaz-Bossini and L. Moreno, "Accessibility to mobile interfaces for older people," *Procedia Computer Science*, vol. 27, pp. 57–66, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050914000106

[36] 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*, ser. DSAI '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 286–293. [Online]. Available: https://doi.org/10.1145/3218585.3218673

[37] "Accessibility for products - principles," https://www.bbc.co.uk/accessibility/forproducts/guides/mobile/principles/, accessed: 2023-09-23.

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

[39] M. Pandey, S. Bondre, S. O'Modhrain, and S. Oney, "Accessibility of ui frameworks and libraries for programmers with visual impairments," in *2022 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2022, pp. 1–10.

[40] K. Kishore, S. Khare, V. Uniyal, and S. Verma, "Performance and stability comparison of react and flutter: Cross-platform application development," in *2022 International Conference on Cyber Resilience (ICCR)*, 2022, pp. 1–4.

[41] H. A. Zahra and S. Zein, "A systematic comparison between flutter and react native from automation testing perspective," in *2022 International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2022, pp. 6–12.

[42] M. Toomim, A. Begel, and S. Graham, "Managing duplicated code with linked editing," in *2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 2004, pp. 173–180.

[43] L. Ardito, L. Barbato, R. Coppola, and M. Valsesia, "Evaluation of rust code verbosity, understandability and complexity," *PeerJ Computer Science*, vol. 7, p. e406, Feb. 2021. [Online]. Available: https://doi.org/10.7717/peerj-cs.406

[44] M. Flauzino, J. Veríssimo, R. Terra, E. Cirilo, V. H. S. Durelli, and R. S. Durelli, "Are you still smelling it? a comparative study between java and kotlin language," ser. SBCARS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 23–32. [Online]. Available: https://doi.org/10.1145/3267183.3267186