FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

User-Centered Classification of iOS Vision Accessibility Problems

Diogo Pinto Soares de Melo

Mestrado em Engenharia de Software

Supervisor: Professora Ana Cristina Ramada Paiva

July 19, 2021

User-Centered Classification of iOS Vision Accessibility Problems

Diogo Pinto Soares de Melo

Mestrado em Engenharia de Software

Approved in oral examination by the committee:

Chair: Professor Nuno Honório Rodrigues Flores
External Examiner: Professor Alberto Manuel Rodrigues da Silva
Supervisor: Professora Ana Cristina Ramada Paiva

July 19, 2021

Abstract

Ever-growing access to smartphones nowadays has drastically increased the importance of accessibility in mobile apps. Such is reinforced by the legislative accessibility acts implemented in the EU and USA. One out of seven people have a disability, and a quarter of the disabled people are visually impaired. VoiceOver, the screen reader incorporated on Apple's devices, has been awarded by the American Foundation for the Blind for the groundbreaking impact of its iOS implementation. In fact, accessible design drives innovation and improves overall user experience and satisfaction, regardless of disabilities or social aspects.

Most studies on the subject investigate how accessibility problems reported by users are covered by accessibility guidelines. However, relating reported problems with the available guidelines requires expertise and they do not take into account every problem reported by disabled users, failing to promote the comprehension of such problems. Thus, developers and testers do not find them clear nor easy to understand, leading to social issues regarding accessibility, that, in turn, prompt a minor focus on accessibility and a lower priority in fixing these issues. Researchers have shown that, given the scarceness of accessibility focused reviews in official app stores, most investigations are based on individual interviews, differing significantly from how disabled users usually report accessibility bugs - feedback forms, email or social media. The AppleVis Bug Tracker and App Directory include user focused bug descriptions and user submitted app reviews concerning vision accessibility on iOS.

This investigation proposes an user-centred classification of iOS vision accessibility bugs. Resulting from an analysis of the data in the AppleVis Bug Tracker and based on the terminology employed by visually impaired users, this classification typifies all the iOS vision accessibility bugs analyzed and is also structured according to the available accessibility guidelines.

The classification was applied to a subset of inaccessible app reviews from AppleVis' iOS App Directory, effectively covering vision accessibility errors reported by users in iOS apps. It was also the subject of a questionnaire targeting screen reader users. In fact, we are 95% confident that the majority of visually impaired users will regard the proposed classification as intuitive.

This investigation deepens the state of the art by scientifically validating the completeness and intuitiveness of a classification based on both the terminology and written means employed by visually impaired iOS users to describe bugs, and also conforming to the available accessibility guidelines. Therefore, we hope to contribute not only to the developers and testers' comprehension of bugs, facilitating their job of relating them with those guidelines, but also to users, once fixing the accessibility bugs they face becomes faster and easier.

Keywords: Accessibility, Mobile Accessibility, Vision Accessibility, iOS, iOS Bugs, Accessibility Bugs, Bugs Classification, Classification

Resumo

O crescente acesso a *smartphones* hoje em dia aumentou a importância da acessibilidade em aplicações móveis. Tal é reforçado pela legislação vigente na UE e nos EUA. 1 em cada 7 pessoas tem deficiência e um quarto destas são deficientes visuais. O VoiceOver, leitor de ecrã incorporado nos dispositivos da Apple, foi premiado pela *American Foundation for the Blind* pelo seu impacto inovador em iOS. De facto, o design acessível impulsiona a inovação e melhora a experiência e satisfação do utilizador, independentemente de deficiências ou aspectos sociais.

Os estudos na área tendem a investigar como os problemas de acessibilidade reportados são cobertos pelos *standards*. No entanto, relacionar os problemas reportados com os *standards* disponíveis requer especialização e estes não englobam todos os problemas reportados - não contribuindo para a sua compreensão. Assim, os *developers* e os *testers* não os consideram claros, levando a problemas sociais, que, por sua vez, levam a um menor foco na acessibilidade. Os investigadores demonstram que, dada a escassez de análises focadas em acessibilidade nas *app stores* oficiais, a maioria das investigações são baseadas em entrevistas individuais, diferindo significativamente de como utilizadores com deficiência geralmente reportam bugs de acessibilidade - formulários de *feedback*, *e-mail* ou redes sociais. O Bug Tracker e o App Directory do AppleVis incluem descrições de bugs focadas no utilizador e *app reviews* submetidas por utilizadores sobre acessibilidade visual em iOS.

Esta investigação propõe uma classificação de bugs de acessibilidade visual em iOS centrada no utilizador. Resultante de uma análise dos dados do Bug Tracker da AppleVis e baseada na terminologia empregue por deficientes visuais, esta classificação tipifica todos os bugs de acessibilidade visual de iOS analisados, estando também estruturada de acordo com os *standards* de acessibilidade disponíveis.

A classificação foi aplicada a um subconjunto de *reviews* de *apps* consideradas inacessíveis do App Directory de iOS da AppleVis, cobrindo efetivamente os erros de acessibilidade visual reportados por utilizadores em *apps* iOS. Foi também aplicado um questionário a utilizadores de leitor de ecrã. De facto, prevemos, com 95% de confiança, que a maioria dos utilizadores com deficiência visual considerará a classificação proposta intuitiva.

Esta investigação aprofunda o estado da arte ao validar cientificamente a classificação proposta como completa e intuitiva. Trata-se de uma classificação baseada na terminologia e nos meios escritos empregues por utilizadores de iOS com deficiência visual para descrever bugs. Simultaneamente, encontra-se em conformidade com os *standards* de acessibilidade disponíveis. Por conseguinte, esperamos contribuir não apenas para a compreensão dos bugs por parte de *developers* e *testers*, facilitando o trabalho de os relacionar com esses *standards*, mas também para os utilizadores, uma vez que corrigir os bugs de acessibilidade que enfrentam se torna mais rápido e fácil.

Palavras-Chave: Acessibilidade, Acessibilidade em Dispositivos Móveis, Acessibilidade Visual, iOS, Bugs em iOS, Bugs de Acessibilidade, Classificação de Bugs, Classificação

Acknowledgements

To my parents and sister, for their support throughout this long and demanding chapter of my life, my most felt "thank you".

I am grateful beyond words to Inês for the help and reassurance, as well as all the unexpected laughters in the most stressful times. My profound appreciation to Catarina, for being a comrade in this insane experience of working and studying simultaneously. Without her, I would never have concluded this degree. I am truly thankful for all the amazing friends and family I have, and with whom I didn't share the laughs, talks, walks and plans I desired for this 2 tiresome years. Also, thanks to my MESW colleagues and friends who eased its hardest moments.

I want to thank my supervisor, Professor Ana Paiva, for accepting the challenge of guiding me through this work, even though in an unfamiliar investigation area, and for the dedication and trust she put in me. I would also like to express a special recognition to the professors and staff members at FEUP's MESW. As a blind person, it has been a challenging but rewarding journey to become a Software Engineering at FEUP.

My deepest gratitude to Professor Paulo Teles, who taught me a decade ago in FEP, for reviving my knowledge of statistics after all these years.

I would also like to acknowledge Critical Software for the incentive to enrole in this degree, as well as to my work colleagues for their support in its most stressful moments, there and, more recently, at Blip.

I also want to thank my guide-dog Fiona for her guidance and patience during countless hours at FEUP.

Diogo Melo

"Pursue happiness with diligence."

Mastodon. (2011). The Sparrow. On *The Hunter*. Roadrunner Records.

Contents

List of Tables

Abbreviations

API	Application Programming Interface
BBC	British Broadcasting Corporation
EU	European Union
IDE	Integrated Development Environment
KIF	Keep It Functional
UI	User Interface
USA	United States of America
WAI	Web Accessibility Initiative
WCAG	Web Content Accessibility Guidelines
WHO	World Health Organization

Chapter 1

Introduction

The present research work addresses the domain of Software Accessibility. The current introductory chapter overviews the context and motivation of this investigation, introduces the problem, defines the objective and presents the overall structure of the document.

1.1 Context and Motivation

According to the World Health Organization (WHO) [45], more than 1 billion people live with a disability, representing 15% of the global population [32]. There are various possible subdivisions of this vast and heterogeneous group, with the most common comprising 5 subgroups: auditory; cognitive, learning and neurological; physical; speech; and visual disabilities [15]. As such, worldwide approximately one out of every seven people has a disability or impairment, which affects how they interact with the world and technology [8]. Within this 1 billion disabled people, WHO [45] estimates that 285 million have visual impairments, of whom 39 million are blind. Demographic changes also point to a significant growth in the elderly population [26]. Thus, addressing the accessibility needs of this ever growing and increasingly more technological inclined group becomes of utmost importance [30].

Accessibility can be defined as a means or a facilitator of the full integration of all groups in society, regardless of their disability. This goal of inclusion reinforces the duty to create and adapt environments to allow access to all their aspects, for all intervenients, on an equal basis [30, 32]. Therefore, inclusion is considered a basic human right regardless of people's impairments [2, 24].

Usability is a concept that is generally related with accessibility. It focuses on a product's quality and ease of use. Universal design is also frequently employed in the context of accessibility and usability, as it preconizes the design of products and services to be usable by all people, to the greater extent possible [26].

The importance of accessibility in mobile apps and websites has drastically increased with the growing usage of smartphones as a means of communication, education, entertainment, information and business [2]).

The European Union (EU) adopted the European Accessibility Act in 2019, in order to set common accessibility requirements in all EU member states. The European Accessibility Act imposed that, by June 23, 2021, all new and existing mobile apps had to conform to the accessibility directives proposed [21]. However, these directives regarding mobile apps lack clarity. In fact, besides to WAI [43] initiatives still in development, they also point to WCAG 2.0 and WCAG 2.1, that do not encompass all the mobile related guidelines [21, 37, 41]. The United States' Americans with Disabilities Act also enforces accessibility on web and mobile apps [17].

Although the objective of these accessibility acts is to promote accessibility and nudge developers into improve their knowledge on the topic, some brands have opted for third-party solutions that promise to automatically scan and reformat web pages to ensure they are accessible and able to work with assistive technologies without requiring extra work from companies and their developers. Nonetheless, blind advocates claim that these solutions fail to make websites accessible, even breaking accessibility in some cases, therefore urging brands to avoid website overlays due to their inability to deliver the claimed accessibility. The deployment of this overlays, as "cheap, fake short-cut" solutions, illustrates the lack of accessibility expertise in companies and the problems that disabled users still face, damaging brands' reputation [4].

Apple's screen reader, Voiceover, has been acclaimed as a groundbreaking innovation that improves the quality of life of blind people. Moreover, iOS is considered the most accessible mobile platform in the market. As a result, Apple has received several awards from the American Foundation for the Blind for its positive impact [22]. Regardless of the recognition and awards inherent to this pledge to accessibility, Apple has deepened its commitment to this and other core values for 2021 [20].

Nonetheless, there are accessibility problems with iOS itself and its native and third-party apps. AppleVis, an online resource for blind and low vision users of Apple devices [14], provides a list of active and fixed iOS bugs and an app directory with accessibility focused iOS app reviews [12, 13]. In this repositories several accessibility problems in iOS, native apps and developer apps can be scrutinized.

Beyond the aforesaid aspects, personal preference and my experience as a daily iOS and macOS user for the past 8 years influenced the choice for exploring Apple's mobile operating system on this investigation. Vision accessibility on iOS is of utmost importance in my life due to the independence I feel as a blind user since I started to enjoy the accessibility provided by VoiceOver. After the impact I experienced by having accessibility incorporated on my devices, my career path changed to informatics and researching this topic represents the confirmation of this change.

1.2 Problem

There are various studies regarding accessibility focused on accessibility barriers faced by visually impaired users. The most common objective is to investigate how WCAG guidelines cover accessibility problems reported by users [3, 31].

However, not all problems reported by disabled users are taken into account in those guidelines [3, 19] and relating reported problems with them requires in depth knowledge of the guidelines, supporting documentation and prior experience [19]. There are also tools to automatically verify the accessibility of applications. Nonetheless, they only cover a low percentage of the available guidelines [31, 37]. Furthermore, developers and testers do not find the guidelines clear nor easy to understand [38].

The literature identified the need of a more user-centered approach to ensure that the many problems faced by disabled users are taken into account and given the proper relevance [19].

Such need arises from the mismatching between the natural language employed by users to report accessibility problems and the available guidelines. This mismatching further enhances the difficulty of mapping problems reported by users with the guidelines, and those guidelines often do not contribute to understand them [19].

Moreover, the lack of accessibility focused reviews in official app stores, due to the low percentage of disabled users [5], leads to investigations based on detailed individual interviews [3, 19], which is not representative of how disabled users usually report accessibility bugs - feedback forms, email or social media.

Consequently, when facing accessibility problems reported by users, and upon the absence of resources to understand them, developers and testers get confused [19], leading to social issues regarding accessibility [38]. In turn, this leads to a minor focus on accessibility [38] and a lower priority in fixing accessibility issues [19].

1.3 Objective

Our thesis hypothesizes that we can propose an user-centered classification able to typify iOS bugs reported by visually impaired users, in conformity with the guidelines available and the language employed by users.

The aim of this work is to create an user-centered classification of iOS vision accessibility bugs that is complete and intuitive.

This classification must be based on the language employed by users to report such bugs and also structured in conformity to the available accessibility guidelines. To do so, the data available at the AppleVis iOS Bug Tracker [13] and iOS App Directory [12], as well as WCAG [41, 42] and BBC [16] guidelines, will be explored.

Hence, this classification can contribute to enhance the comprehensibility of iOS vision accessibility problems, promoting their easier and faster fixing, therefore ensuring usability for all users.

1.4 Structure

Chapter 2 presents the state of the art regarding software accessibility, focusing on mobile and vision accessibility.

The Background Chapter 3 describes VoiceOver concepts and presents AppleVis [14] and its repositories.

In Chapter 4, the proposed classification and the process that led to its construction are explained.

The penultimate Chapter, 5, covers the validation phase, comprising the assessment of the classification's completeness and intuitiveness.

Finally, Chapter 6 presents the conclusions of this investigation, its practical and theoretical contributions, the limitations of the study as well as suggestions for future work.

Chapter 2

State of the Art

Software accessibility is addressed on diverse research papers. Several sets of keywords were constructed by combining the terms accessibility or accessible with various other words (e.g. software, technology, iOS, mobile, automation, checking, properties, guidelines, testing, processes, etc). These sets were searched in specific and general search engines, such as Google Scholar, Scopus and Engineering Village.

The investigations explored focus in three main areas: guidelines and best practices; improvements to software development processes; and testing tools to automatically detect accessibility bugs.

2.1 Guidelines

The most common accessibility guidelines mentioned in the literature are the WCAG guidelines and, although not as often referred, the BBC Mobile Accessibility Guidelines have also been referenced [37].

2.1.1 WCAG Mobile Guidelines

The W3C Web Accessibility Initiative (WAI) is finishing the 2.2 version of the Web Content Accessibility Guidelines (WCAG), which comprises, amongst others, new success criteria addressing needs of users of mobile devices [43].

For as long as this new version is not finalized, there are available WAI guidelines describing the application of the WCAG 2.0 and its principles, guidelines, and success criteria to web and non-web mobile applications and content [41]. These recommendations take into account mobile web content, mobile web apps, native apps, and hybrid apps using web components inside native apps. They also include the Mobile Web Best Practices and the Mobile Web Application Best Practices, offering general guidance to developers regarding the accessible creation of applications and content targeting mobile devices [41, 43].

These guidelines are marked as a draft version since February 2015 and receiving feedback as an essential component of its success. Some of these guidelines and associated success criteria have already been incorporated into WCAG 2.1 [42], published on June 2018. However, the document comprising these mobile guidelines presents itself as based on WCAG 2.0 [41]. It mirrors its four principles [40], which can be summarized as:

I "Perceivable" – Information and user interface components must be presentable to users in ways they can perceive – This principle encompasses:

- Providing text alternatives for non-text content, that can be changed into other forms people need (e.g. large print, braille or speech);
- Alternatives for time-based media;
- Creating content presentable in different ways without losing information or structure;
- Presenting distinguishable visual and audio content, clearly separating foreground from background;

II "Operable" – User interface components and navigation must be operable – This principle contains:

- Making all functionality accessible using a keyboard;
- Allowing users enough time to read and use the content;
- Avoid content designs known to cause seizures;
- Help users in navigating and finding content, as well as in determining where they are in the page.

III "Understandable" – Information and the operation of user interface must be understandable – This principle includes:

- Making the text content readable and understandable;
- Making the contents appear and behave in predictable ways;
- Helping users avoid and correct mistakes, such as in filling forms.

IV "Robust" – Content must be robust enough that it can be interpreted reliably by a wide variety of user agents, including assistive technologies – This principle is solely composed of:

- Compatibility of the content with current and future user agents, including assistive technologies, such as screen readers.

The recommendations from the WCAG mobile guidelines [41], grouped by the four previously presented categories, are the following:

I "Perceivable" – Take into account the smaller screen size of mobile devices, considering the available space instead of the high resolution to decide the amount of information to present in a usable way. This recommendation also refers to the usage of screen magnification in case of low vision users. The encouraged best practices comprise providing a dedicated mobile version to smaller screens and implementing a responsive design, adapting the content to have fewer modules, fewer images, or to focus on important mobile usage scenarios, as well as collapsible navigation menus, touch controls with reasonable sizes and positioning form fields below their label in portrait orientation.

Some recommendations regarding zoom, both to a wider audience and to assistive technology users, are also defined. The accessibility features taken into account in the article include OS level features, such as the definition of the default text size, that can be found on iOS' Display Settings, and the zoom accessibility feature, that allows low vision users to magnify the entire screen and horizontally and vertically pan around it to navigate the magnified content. It also includes browser features, namely the definition of a default text size of the text rendered in the browser's viewport, the availability of reading mode, that allows the user to see the main content at a pre-defined text size, and the usage of the pinch to zoom gesture. The success criteria mention the requirement of text to be magnifiable to 200% of its size using assistive technology, content not deterring users from magnifying it and the inclusion of techniques that allow the text size increasing by the system defined settings (e.g. display size defined at the operating system level) or by users through on-page controls.

Once mobile devices are used in very different environments, such as outdoors and other venues where natural or artificial strong light sources can cause glare, the document advocates for the definition of good contrast in the user interfaces. Particularly, its aim is to minimize the challenges that low vision users may face in such environments. The suggestions regarding contrast comprise some ambiguous requirements, with the minimum text contrast ratio ranging from 4.5:1 to 7:1 or from 3:1 to 4.5:1, for large-scale text. The exhortation for developers is to use the lesser ratio only when text is equivalent to 1.2 times bold or 1.5 times the size of default platform text.

II "Operable" – Although modern mobile devices are almost exclusively devoid of physical keyboards, the use of external physical keyboards is common by some disabled users, mainly users with vision and motor disabilities. Therefore, the recommendations encompass proper keyboard accessibility and control of applications.

Some directions are made regarding touch targets' size and distance between elements, namely the definition of a minimum height and width of 9 millimeters and the existent of a small inactive space surrounding smaller elements.

The touch gesture based navigation of modern mobile operating systems, ranging from simple taps to very complex multiple finger gestures and multiple taps and drawn shapes, can cause problems or be impossible to use by users with disabilities. This particularly affects

screen reader users, when interaction modes are based on two-step processes for focusing and activating elements instead of direct touch manipulation, and also users with motor or dexterity impairments or who rely on devices in which multi touch gestures are impossible to perform (e.g. head pointers or styluses). In some mobile operating systems, it is possible to simulate complex gestures with simpler ones using on-screen menus (e.g. iOS' Accessibility Action). Recommendations in this matter involve giving the users the opportunity to cancel an action before committing to it, by moving the cursor (or finger) to outside the element. Another issue presented is the discoverability of such gestures, with the suggestion of having some indicator to prompt the user to execute the necessary gesture to activate a functionality.

Similarly, there are other gestures that entail physical device manipulation, such as shakes and tilts of the mobile device, which may be challenging or impossible for users who have difficulty or are unable to hold a mobile device.

Although some operating systems allow on-screen menus to perform those actions (e.g. iOS's Assistive Touch), it is recommended that developers provide touch and keyboard alternative controls to features that rely on device manipulation. Similarly to the previous point, discoverability is another problem of these device manipulation actions.

These guidelines also commend the placement of buttons on easily accessible screen areas regardless of device orientation and positioning. Some mobile operating systems allow the user to temporarily shift the display down or sideways for better screen access (e.g. iOS's Reachability on bigger iPhones), but it is recommended that layouts are designed with right and left handed usage and different thumb range of motions in mind.

III "Understandable" – Applications should support all available screen orientations and inform assistive technologies about device orientation changes. This is particularly important to users who can't rotate the screen to match orientation changes, such as users with their smart phone mounted on the arm of a power wheelchair. Screen reader users also need to be informed of screen orientation changes, because navigation gestures are screen orientation sensitive and an incorrect gesture may be performed if the orientation is different than the one perceived by the user.

Applications and pages should exhibit a consistent layout between screens and orientations in components repeated throughout these different screens and orientations, which means respecting the relative orders in which elements are sequenced in the various contexts in which they appear. This requirement goes beyond WCAG 2.0 and relates to consistent navigation and consistent identification.

Important page elements must be presented before scrolling down the page, as the small screen on mobile devices limits the available information displayed in each page and some users may only visualize one part of the screen at once (e.g. low vision users relying on magnification).

Grouping elements that perform similar actions is also advocated. For example, a link icon with a link text, both performing the same action or going to the same destination, should be contained within the same actionable element. This increases the touch target size and reduces the number of redundant focus targets, improving the experience for dexterity impaired, external keyboard, switch control and screen reader users.

The clear indication of items' actionability is also crucial to all users, disabled or not. These elements (e.g. buttons, links) should be obviously distinguishable from elements that do not trigger any change of action. By following existing interface design conventions, developers can visually convey the actionability of these trigger elements by their shape, color, style, positioning, label, and iconography. This information shall also be programmatically announced to users of screen reading technologies (e.g. through Accessibility Traits on iOS). This guideline is also an addition since WCAG 2.0, as part of the consistent navigation and consistent identification success criteria.

As previously mentioned, the usage of complex touch gestures and device manipulation gestures can cause problems to some users. Besides performing the gestures themselves, the discoverability of those gestures and their accessible alternatives shall also be taken into account. Instructions, such as overlays, tooltips, tutorials and others, must be provided to detail and explain the available gestures and accessible alternatives. Although these instructions shall be prominent on first use, they should always be easily accessible to all users.

IV "Robust" – The definition of alternative keyboards to the data type of the text field being entered can be of assistance to users by avoiding mistakes and entering valid data by default. Nonetheless, the common keyboard layouts shall be respected, since screen reader users may feel confused with drastically different virtual keyboard layouts.

Reducing the amount of data entry the users need to do can also make their usage of an app or page easier. This can be accomplished by providing select menus, radio buttons, check boxes or by automatically entering known information, like date, time or location.

Supporting the accessibility features of each platform is also recommended, once most of the work is already implemented by the operating system and only small programatic adjustments may be needed.

2.1.2 BBC Mobile Accessibility Guidelines

The British Broadcasting Corporation (BBC), the British public broadcaster, has also defined a set of independent standards, the BBC Mobile Accessibility Guidelines [16, 37]. These guidelines [16] are divided on eleven high-level topics, preceded by overall principles and followed by some particular recommendations. The guidelines comprise technologically achievable best practices, considering current mobile assistive technology, and are easily testable with specific objective criteria - marked as "must" or "must not" - as well as other less testable recommendations that,

nonetheless, are of high importance in the accessibility of mobile websites and apps - marked as "should" or "should not".

The three general principles encourage developers to use the platform and web standards as they are intended to be used, base their apps on standard interface controls as much as possible and support the specific platform accessibility features.

I "Audio and Video" – BBC exhorts the implementation of alternatives for embedded audio and visual content, such as subtitles, sign language, audio description and transcripts. It equally suggests that audio content must not play automatically, unless a button to pause, stop or mute the audio is present and easily reachable or if the user is informed, by the app or its natural flow, that audio content will be played.

It is also recommended that relevant metadata is available for all media, that background music, ambient sounds, narrative and editorially significant sound effects have separated volume controls and, in games or interactive media, that narrative audio does not overlap or conflict with native assistive technology.

II "Design" – BBC advocates that the text and its background must have sufficient contrast, that the meaning must not be conveyed solely based on color, that the main content must still be accessible when styling is either not supported or has been removed, and that touch targets must be large enough to being accurately pressed.

The spacing between actionable elements, the consistency of the users' experience and adjustability of media by users based on their ability and preference are also suggested. Other must success criteria are the capability of users to control font sizing and user interface scaling - possible using operating system features; the clear distinguishability of actionable items - such as links or buttons; and their visual state change when focused.

This list of design aspects also encompasses interfaces providing multiple ways to interact with content and the prohibition of content to visibly or intentionally flicker or flash more than three times per second.

III "Editorial" – It is advised that element labeling should be consistent both within applications and websites on their own and also across websites and native applications and, when needed, applications should provide additional instructions to supplement visual and audio cues. The active language on an application or page must be specified, and changes in the active language must be indicated.

IV "Focus" – BBC urges developers to make all interactive elements focusable and the inactive elements not focusable, ensure that screen reader focus does not stay trapped in a single element and all temporary screen elements (e.g. on-screen keyboards) can be dismissed and also, to order the content in a logical sequence.

The exhortation extends to a logical and meaningful sequence in the navigation of the actionable content, focus or context not automatically changing during user input, actions only

being triggered when the user undoubtedly commits to it in their specific input method and the support of these alternative input methods.

V "Forms" – The document reinforces that all form controls must be labelled and that the label must be close to the form control to which it refers and laid out appropriately. Controls, labels and other form elements must be properly grouped and a default input format per form field must be indicated and supported by the application or page.

VI "Images" – BBC advices that images containing relevant information or contextual meaning must have an additional accessible alternative (e.g. accessibility label) and that images of text (inaccessible to screen readers) should be avoided.

VII "Links" – The document states that for each link and navigation text there must be an explicit description of the correspondent target or function. Links to alternative formats must also indicate that an alternative is opening and when repeated links point to the same resource they must be combined within a single link.

VIII "Notifications" – BBC suggests that notifications must be both visual and audio in order to be inclusive and that, when available and appropriate, standard operating system notifications should be used. When an error occurs, the advice is to provide clear error messages and, when a non-critical error occurs frequently, feedback or assistance should be provided.

IX "Scripts and Dynamic Content" – The guidelines reinforce that applications and websites must be built to work in a progressive manner with the goal of providing a functional experience for all users (e.g. supporting previous operating system versions), that pages must not be refreshed without warning, once the focus of a screen reader loses its position in such refreshes, and that time outs must be adjustable to the time needed for each user to conclude a given task.

Developers may also make available a pause, stop or hide control where either constantly updating Media or animated content is present and support the adaptation of input controls.

X "Structure" – The document exhorts that each page and screen must have a unique title to be clearly and uniquely identifiable, that the content must provide a logical and hierarchical heading structure (essential to an efficient screen reader navigation) and that controls, objects and grouped interface elements must be represented as a single accessible component. It also is suggested, if supported by the mobile platform, that containers and landmarks should be used to describe page and screen structure.

XI "Text Equivalents" – The guidelines urge applications and pages to provide alternatives to non-textual context (e.g. images, graphics, diagrams) that briefly describe the editorial intent or purpose of such elements, as well as hiding from assistive technology decorative images which content is irrelevant to the context (e.g. backgrounds).

It is also encouraged that tooltips do not repeat information already contained in the element they refer to, that elements must have accessibility properties set appropriately and that meaning must not be conveyed by visual formatting only.

The BBC Mobile Accessibility Guidelines close with some concrete recommendations, encouraging developers to offer a core accessible website, use progressive enhancement, link mobile and desktop sites and minimize text fields.

Despite the vastness and detail of the BBC Mobile Accessibility Guidelines [16], there are some recommendations missing when compared to the WCAG mobile guidelines [37, 41].

The design section is missing the WCAG indications regarding the minimization of the amount of information on each page compared to the equivalent desktop or laptop version, the positioning of important information to make it visible without requiring scrolling and providing labels or instructions when content requires user input.

The forms section also does not mention the position of form fields below their labels instead of beside them.

BBC's guidelines do not mention specifically the ability to perform all functionality through a keyboard interface [37], although it can be argued that the requirement of supporting various input methods incorporates this WCAG requirement [16].

However, these guidelines do not take into account the complexity of multi touch gestures and WCAG's recommendation to make those as simple as possible and to provide simpler alternative controls to assistive technology users, as well as the recommendation to support all screen orientations and to inform programmatically about orientation changes to assistive technology users [41].

2.1.3 Guideline Coverage

The studies that focus on guidelines analyze how thoroughly the available guidelines, usually WCAG, can cover accessibility bugs reported by users. In her most recent study, Alajarmeh [3] asked sixteen visually impaired users from the USA to navigate apps and websites on their mobile devices to evaluate how WCAG 2.1 guidelines covered the problems found. From the 34 identified accessibility problems, 8 were not covered by WCAG 2.1 guidelines. Although WCAG 2.1 represents a positive evolution from WCAG 2.0, the author believes that the conformance level of some success criteria is not adequate to the severity of the bugs that may result from disrespecting those guidelines. The fact that 26 of the problems are covered by WCAG 2.1 guidelines also demonstrates that developers and content providers continue to overlook accessibility practices, as previous studies have also concluded [1, 18, 35].

Clegg-Vinell et al. [19] recognize the importance of guidelines to raise awareness of accessibility and to provide a point of reference during the design, development and verification of a website or app. However, and as explored by Alajarmeh [3], the authors emphasize that relying on guidelines to understand the severity level of problems results in ratings very different than what users perceive. Furthermore, the authors claim that reporting problems alongside guidelines can

prioritize the fix of the reported problem, but linking problems with guidelines often requires the work of an accessibility consultant with in depth knowledge of the guidelines, supporting documentation and prior experience.

A thorough case study was conducted, involving several accessibility consultants and a broad range of disabled users with various disabilities. The users were asked to perform several tasks in a number of websites and apps in different platforms while the moderator monitored the problems arising during the process. The severity of the problem was determined by its frequency, impact and persistence. Then, each problem was classified according to if it was easily mappable to a guideline, if the mapping was possible but only with knowledge from a specialist or if the problem had impact but no mapping in the guidelines. The severity levels of the guidelines were considered inadequate to classify the problems faced by those users and some of these reported issues were, in deed, considered a significant barrier to more than one disability group. Clegg-Vinell et al. [19] conclude that problems reported by users often do not relate to guidelines, which makes the developers' job of understanding said problems harder and more time-consuming.

Therefore, the authors advocate for guidelines that embrace a more user-centered approach to ensure that problems with impact on users are given the proper relevance, as well as to incorporate issues that have not been taken into account [19].

2.2 Processes

The research about developer processes introduces accessibility focused processes into teams to evaluate their impact on the accessibility of the product developed.

Sánchez-Gordón and Moreno [39] proposed a proof of concept of the test process on the requirements phase of accessibility development lifecycle, based on a proposal by Microsoft, defining how accessibility is incorporated into each phase of the software development process. This later has been adapted to accessible Web development on an Agile environment [34].

On the requirements stage, engineers may generate personas that represent users with various degrees and types of disabilities, create scenarios to define which design features could help those users and illustrate how they will accomplish tasks by interacting with the product in an accessible manner. Then each feature shall be prioritized and, in the end, all users must be able to complete the proposed use cases.

On the testing stage, the proposed test process includes: during the planning and control of tests, preparing the test strategy to include the satisfaction of accessibility standards; in the analysis and design phase, deciding which and to what degree the accessibility tests should be automated and which accessibility standards and guidelines are incorporated into the tests' exit criteria; in the implementation and execution phase, creating the test specifications to evaluate the satisfaction of accessibility standards, evaluate the exit criteria and report them and the verification and validation of the incorporation of accessibility guidelines and standards in the test plan and; finally, the test closure phase incorporates the test summary activities from the testing metrics and lessons learned.

Steen-Hansen and Fagernes [38] proposed a series of processual guidelines in order to incorporate accessibility into the entire development and testing process of web applications, which can be directly transposed to the development process of mobile applications. These guidelines aim to make accessibility a natural part of the development process, not by certifying that the final product meets certain standards or guidelines but making a statement that product development process was based on an approach that has taken into account educated decisions regarding the accessibility of the product.

The authors identified two main problem areas in the introduction of accessibility into development processes, namely social and tool issues. Social issues concern the people involved in the project and their priorities, expertise and pre-assumptions, while tool issues comprise the availability and ease of use of tools to test and ensure accessibility [38].

The collective notion of accessibility's minor importance is a common social issue amongst development teams, management and clients, which can lead to lack of time and funding to focus on the product's accessibility. This pushes accessibility to the end of the development process, turning it into an after-thought.

The vastness of the accessibility field, as presented priorly, and the lack of knowledge and education in this area is also a social issue. Accessibility is considered an interdisciplinary field that ranges from good coding, solid design practices, knowledge of specific tools to automate accessibility testing and usability testing with disabled users. This vastness may overwhelm some developers, leading to negative feelings about accessibility and its complexity and learning curve. In turn, managers see it an expensive investment and postpone it until an accessibility-minded client appears and catalyzes the learning process.

Moreover, a bigger problem is the fact that some people consider accessibility an unnecessary and inappropriate endeavor, with some even considering it an intrusion to their aesthetic and graphical design sensibilities. To solve this social issue, it is vital to enlighten developers, managers and clients about the importance of accessibility and in what it consists.

Using tools and guidelines for manual and automatically test the accessibility of a product can be useful, specially to help developers getting familiarized due to never before having had contact with accessibility or not having a very deep knowledge about the topic.

However, tools and guidelines can be difficult to comprehend and time-consuming, and may require accessibility expertise in order to be fully and effectively used. Additionally, despite being helpful in finding the accessibility problems, tools may be unclear and confusing in the messages they provide to describe and explain the accessibility issue and help the developer to fix it. Hence, the size of the guidelines and their ambiguous nature may be a problem to developers, who favor faster software development tools.

The proposed co-dependent process-oriented development guidelines to facilitate the development of a more accessible product are as follows:

- Have Accessibility Expertise on the Team – Due to the complexity and vastness of the theme, it should comprise at least one person with accessibility experience regarding knowledge of legal requirements, good coding practices and more and less accessible technologies

and frameworks as well as familiarity with accessibility testing tools.

It is recommended that this person has experience doing usability testing with users that require accessibility, knows how to argue with the team, management and client about the importance of accessibility and is involved in the development process since the beginning to avoid using less accessible technologies.

If no such person is available, the team should hire an accessibility expert to help in critical moments of the project, such as the initial planning and design phase, to guide the team in making accessible technological choices and in how to test for accessibility and to be available if any further question arises.

- Communicate Accessibility within the Team – The knowledge of the accessibility expert, regardless of being internal or external to the team, is not useful if it is not shared with the team. The team should communicate how to work to ensure accessibility, ask questions and hear the accessibility expert and share the responsibility of making an accessible product.

  To do that, the management shall allow time to accessibility related meetings and discussions and for the communication of accessibility requirements before the implementation phase.

- Follow Existing Design Principles – As previously stated, accessibility and usability are interrelated properties of software, therefore following known usability patterns, regarding good design practices, well-written code, solid architecture, accessible form validation and error recovery, can make software more accessible due to being more usable to everyone. This allows the team to leverage on existing design and usability knowledge and combine it with the knowledge of the accessibility expert, by designing the interface and deciding on accessibility requirements in parallel.

- Test Accessibility at Key Stages – The importance of testing has been emphasised in both literature and practice. Therefore, to ensure a properly accessible product there should be a properly defined accessibility testing process since the beginning of the project.

  The team should ensure that accessibility testing is performed at key stages in the product cycle, such as when some new core feature has been designed or implemented, and the testing should be automated and focused on small functionalities per test case.

  The management should plan time for accessibility testing and the accessibility expert should be responsible for the testing itself, deciding what to test and how to test it, taking care of not generating false positives and to automate testing as much as possible, disseminating the knowledge to the rest of the team.

  The tests should also encompass actual usability testing with disabled users whenever possible, as well as disability simulation by the team members, such as trying to use a screen reader, using only a keyboard, disabling color and text styles or experimenting physical

simulations of disability (e.g. Cambridge Simulation Glasses) [15]. These guidelines incorporate Sanchez-Gordon and Moreno's [39] test process on the requirements phase of accessibility development lifecycle.

- Introduce Accessibility from the Beginning – This is the main pillar in which all this process is based, as shown in the previous four processual guidelines. Finding accessibility workarounds and fixes is harder and more time-consuming than integrating accessibility in the product design, implementation and testing from the beginning. Choosing the right and more accessible frameworks and technologies from the beginning ensures an effortlessly more accessible product than than a need to change technology throughout or at the end of the project if only then it becomes clear that users will not be able to use it. Accessibility metrics should also be decided since the beginning of the project.

Luján-Mora, firstly with Masri [30] and then with Sanchez-Gordon [34], proposed the integration of accessibility on Agile methods that validates these processual guidelines. Such proposals were centered on web accessibility, but they can directly be extrapolated to mobile accessibility. Due to the Agile focus on interactions and collaborations between the different stakeholders and their engagement with team members, it is suggested that Agile's user-centered design needs to incorporate accessibility user testing to meet the requirements of the stakeholders that rely on accessibility to use the product [30]. Since Agile's primary measure of progress is working software, and it is characterized by frequent deliveries, measured in weeks rather than in months, accessibility testing should be performed since the beginning of the project, instead of being postponed to its end. These tests should involve both user accessibility testing, performed frequently, and the automation and continuously run of accessibility tests, in order to constantly detect arising issues and ensure their immediate fixing. This approach is suitable to incorporate the testing approach proposed by Sanchez-Gordon and Moreno [39]. This ensures the satisfaction of the final user, regardless of disability, and the capability to quickly react to changes in the product or in the market [30].

Although these approaches are of a theoretical nature, some case studies [6] validate approaches based on the same principles: define a process, a set of tools and guidelines and obtain more accessible results. These results are achieved due to the adopted process in the software development and not based on focusing solely on the final product's accessibility [6, 38].

The foundations of this success can be defined as:

- Ensure that accessibility is a constant concern throughout the development phase;
- Familiarization of developers with accessibility guidelines;
- Definition of accessibility metrics and control of those metrics during the development process;
- Dissemination of expertise in using tools to ensure guidelines are being respected;
- Automating processes and tests.

The choice of technology and the accessibility evaluation tools was also considered an important success factor in the analyzed case studies [6].

A crucial recommendation can be extracted from all the analyzed processes: regardless of how many guidelines are written, how many guideline recommendations are checked and how many automated tests are implemented, usability is always the ultimate goal [6, 15, 30, 33, 34, 37, 38, 39]. Since the success of a product can only be measured when it is used by the customer, the accessibility of a product can only be confirmed or denied when the final user uses that product with their assistive technology, given their accessibility needs.

2.3 Tools

In this section, the tools and frameworks to develop and test the accessibility of iOS applications are scrutinized.

2.3.1 Apple's Solutions

As responsible for the iOS operating system and the platforms needed to develop, test and deploy iOS apps, Apple provides tools and frameworks to test apps and their accessibility.

Until 2014, Objective-C was the exclusive programming language used to develop iOS apps, with UIKit being the only framework to implement the app's user interface. In 2014, Apple developed and posteriorly open sourced Swift, allowing iOS development with both Swift and Objective-C in the same code base. Since 2019, Apple also included SwiftUI as a declarative UI structure design framework supported in iOS development, in parallel with UIKit [9, 29].

Apple also offers Xcode, an Integrated Development Environment (IDE) to support all its platforms and features, that can be freely downloaded and used on its macOS platform. Xcode natively supports XCTest, a test framework also developed by Apple that is by default included in every iOS project's code base, with its own test navigator in the IDE to easily create, configure and run tests. It includes unit, integration, performance and user interface testing, supports continuous integration with Apple's Xcode Server and is compatible with the latest versions of Objective-C and Swift [11].

2.3.2 Accessibility Testing Tools

The following tools and frameworks can be used by iOS testers to test the accessibility of their iOS applications:

- Accessibility Inspector

  Developed by Apple and part of Xcode's application set, this native Mac app inspects each element of an app and displays accessibility information about each element present in the user interface. The analysis can be performed manually by testers, to check accessibility properties in a per element basis, or by activating the Accessibility Verifier, which can identify non-textual elements of the interface lacking an alternative description.

This tool is classified as a dynamic blackbox tool that uses a manual approach to accessibility bug detection [37].

- Mobile Web Accessibility Checker

  This mobile application can evaluate the accessibility of mobile websites on an iOS device. It does so by reporting on design accessibility flaws, such as interface color contrast, and also reports if elements have accessibility labels that allow them to be read by screen reading technology.

  This tool is also considered a dynamic blackbox tool that uses a manual approach to accessibility bug detection [37].

- XCUI

  This native Xcode solution is based on two core technologies, the XCTest framework and Accessibility. Once it is developed by Apple, it offers advantages such as full integration with Apple's continuous integration and continuous deployment solutions (Xcode Server and xcodebuild) and its compatibility is up to date with the latest versions of both Objective-C and Swift. It uses the core Accessibility framework, including a rich set of semantic data about the user interface, to identify and find UI elements and simulate user actions' performance on them [11]. It has no specific implementation to automatically test the compliance with any Accessibility guidelines, but its foundation components can be used by testers to implement accessibility tests on their apps [44].

  It is an automated dynamic black box framework that has been used in some of the tools presented below.

- EarlGrey

  Developed by Google, with native Xcode integration, EarlGrey is a white-box functional UI testing framework for iOS that can perform accessibility property checks on iOS apps using Unit Testing.

  Accessibility checkers can be turned on on a test class basis to verify if any component activated on a test fails to comply with the accessibility properties, automatically raising an exception in such cases [36, 37]. It is capable of detecting several types of accessibility defects, such as the lack of alternative text for non-textual components, duplicate alternative texts, insufficient color contrast, inadequate touch target size, overlapping of actionable elements and clickable spans, like actionable words within a text.

  EarlGrey represents a dynamic white box framework that uses test scripts to accessibility bug detection [37].

- KIF

  KIF, an acronym for Keep It Functional, is an integration test framework that can be integrated into Xcode [27]. Like XCUI [11], it also leverages on the accessibility attributes

that iOS makes available to Voiceover, allowing testers to write tests that simulate real user input with accessibility-checking capabilities. However, due to the usage of undocumented Apple APIs, the KIF team states that, despite safe for testing purposes, KIF tests are not recommended to be committed into production, as it can get the app submission denied by Apple. According to the KIF documentation, this problem is common in most iOS testing frameworks, however this disclaimer was not found in other frameworks analyzed [25]. KIF is capable of detecting the same types of accessibility defects EarlGrey does, such as the absence of alternative text in non-textual components, duplicate alternative texts, lack of sufficient color contrast, problems with touch target size, overlapping of actionable elements and clickable spans [37]. KIF also represents a dynamic white box framework that uses test scripts to accessibility bug detection.

- A11yUITests

  A11yUITests is a Testing library for accessibility based on Apple's XCUI User Interface test framework. It is implemented as an extension to XCTestCase that adds tests for common accessibility issues that can be run as part of an XCUITest suite. It follows WCAG guidelines and Apple's own recommendations on accessibility, also leveraging on the accessibility information available to Voiceover using the app's accessibility tree, turning any well-defined XCUITest into an accessibility test [44]. A11yUITests is able to detect some accessibility defect types similar to what EarlGrey and KIF do [37], such as lack of accessibility labels, duplicated accessibility labels, problems with touch target size and overlapping of actionable elements. This framework, however, goes beyond what EarlGrey and KIF can do, verifying the adequacy of labels following Apple's guidelines, such as nonredundant information about element types in labels, the length of labels and alerting if an image has its name as its accessibility label. Notwithstanding, the A11yUITests developer recognizes some limitations in the tool, like only verifying redundant element types in accessibility labels of buttons and not in other element types, the inability to check if the element type is present as an accessibility trait and some speed problems, as well as the current inability to integrate the tests in Continuous Integration and Continuous Delivery pipelines [44]. Nonetheless, it is a valuable contribution to accessibility testing in iOS going beyond other tools in some regards. It represents an automated dynamic white box framework of accessibility bug detection.

2.3.3 Tools Coverage

The investigation about tools consists in evaluating how thoroughly those tools cover accessibility guidelines, usually WCAG, or accessibility bugs reported by users.

Silva et al. [37] studied the accessibility testing tools available on each mobile operating system and analyzed the extent of BBC and WCAG 2.0 guidelines that they were able to cover. For

Table 2.1: Accessibility Guidelines Covered by Testing Tools on iOS

Tools	Mobile Web Accessibility Checker	Mobile Web Accessibility Checker	EarlGrey	KIF
Color Contrast		X	X	X
Touch Target Size			X	X
Spacing			X	X
Actionable			X	X
Visible		X		
Keyboard		X		
Consistency	X			
Alternatives for Non-Text	X	X	X	X

Adapted from [37]

iOS, the authors identified Accessibility Inspector and Mobile Web Accessibility Checker as dynamic manual blackbox tools. On the other hand, EarlGrey and KIF were identified as dynamic script based automated white box tools that detect accessibility problems on iOS.

Nonetheless, the coverage of guidelines that these tools can perform is below 13%. Mateus et al. [31] analyzed the performance of two automated accessibility evaluation tools, MATE [23] and Accessibility Scanner, with 415 instances of accessibility problems encountered by visually impaired users on four mobile applications. The research found that more than 70% of accessibility problem types could only be detected by users, more than 20% could only be detected by the tools and 6% of them could be identified by both users and tools. These results confirmed the aforementioned recommendation of involving users in accessibility testing throughout the project to find and solve eventual accessibility problems.

Table 2.1 illustrates, amongst the guidelines covered, which tools are capable of detecting if a given guideline is covered.

2.4 Discussion

From the state of the art, we conclude that several aspects converge towards the challenging nature of detecting and understanding accessibility problems, which inherently imposes a need for solutions. The tools to verify accessibility cover only a low percentage of the available guidelines [31, 37] and not all problems reported by disabled users are taken into account in those guidelines [3, 19]. Additionally, relating reported problems with the guidelines requires expertise [19] and developers and testers do not find the guidelines very clear nor easy to understand [38].

Therefore, there is a need of a more user-centered approach to ensure that the many problems faced by disabled users are taken into account and given the proper relevance [19].

Although most disabled users are not experts in accessibility guidelines, they are able to use software in their daily lives. When facing accessibility problems, they report them in their natural language, perhaps influenced by the knowledge dissemination amongst their disability groups (e.g. blind users), and that is the language they employ when reporting to developers bugs they need to be fixed. However, developers are not familiar with the common terms used by accessibility

groups to describe the problems they face and, as aforesaid, mapping the problems to the existing guidelines requires expertise and most times does not contribute to understand the problem [19].

This sets users apart from developers as if they communicated in different languages, which can lead to social issues regarding accessibility [38]. The mismatching of realities between bug reports and guidelines confuses developers [19], consequently leading to a minor focus on accessibility [38] and a lower priority in fixing accessibility issues [19]. Some studies refer the lack of accessibility focused reviews in official app stores due to the low percentage of disabled users [5]. Most of the studies base their investigations on individual interviews with disabled users [3, 19], in which the users can narrate live and even demonstrate thoroughly the accessibility problems they face during the process. Although useful in the context of these studies, this method is not representative of how disabled users usually report accessibility bugs, which commonly is through written text on feedback forms, via email or in social media.

Given these premisses, we propose a user-centered solution that could help developers in understanding accessibility problems reported by users that consists in a classification based on the language employed by users to report such problems, structured around the available accessibility guidelines. This classification can contribute to enhance such problems' comprehensibility, promoting their easier and faster fixing, ensuring usability for all users. The scope of this investigation will be vision accessibility problems and the iOS operative system. This option was based on the shortage of time, the prior existence of a broad range of workable data regarding iOS bugs and the author's own familiarity and experience with vision accessibility on iOS.

Chapter 3

Background

The aim of this chapter is to contextualize the topic of vision accessibility on iOS, to explain how the Zoom magnification feature and, specially, VoiceOver work and to present the repositories that will be used throughout this work.

As a blind person with over 8 years of experience using VoiceOver on iOS, I consider myself an expert. Both in personal and professional matters, I have found myself in a position in which I had to explain and demonstrate VoiceOver countless times. Thus, I employed that knowledge to supply a description of VoiceOver in the following section, aiming to provide a proper context to anyone, regardless of having prior contact with a screen reader or not.

3.1 iOS Vision Accessibility

Apple incorporates a vast set of accessibility features as part of its operating systems, ranging from vision to hearing and physical/motor. Every Apple device comes with this features and users can activate them if needed. The devices also allow the configuration of one or more accessibility features to be quickly triggered when an accessibility shortcut is performed (e.g. triple tap the side button in modern iOS devices or the Home button in old ones, double or triple tap the back of a recent iPhone or press the control key and triple tap the power button in modern Macs).

In iOS, the main vision accessibility features are Zoom and VoiceOver. Due to their complexity, these features change the gestures with which users interact with the iOS device. One could mention other features regarding vision accessibility on iOS, but they are toggles and settings that merely change the system appearance or add functionality, with no impact on the device operation, unlike Zoom and VoiceOver. As examples of these simpler kinds of features in iOS 14 there are adjustments to apply bold formatting to text, increase the text size, reduce transparency, reduce the intensity of bright colors, increase contrast, not rely on color to convey meaning, color filters and invert screen colors (with or without inverting colors of media elements - classic or smart invert,

respectively). Moreover, there are features to decrease the use of motion throughout the user interface, activate audio description when available in media, use the device's camera as a magnifier and read aloud text when desired.

3.1.1 Zoom

Zoom allows the display to be magnified to up to 15 times the content's normal size, in either full screen or window zoom, and the application of color filters while doing so. Besides keyboard shortcuts, Zoom can be controlled with touch gestures as follows:

- Double tap three fingers to activate and deactivate Zoom;
- Drag three fingers around the screen to move the zoomed region - naturally, as the zoom increases and the screen display remains the same size, the visible magnified content is a fraction of the normal user interface. Therefore, the user has to pan the zoom focus around the screen to see all the interface elements;
- Double tap with three fingers and then drag to change the zoom level: the drag gesture must be performed upwards to increase the magnification and downwards to decrease it.

3.1.2 VoiceOver

VoiceOver is a screen reader that allows users to use their iOS device without seeing the display. It provides feedback in a digital voice of what is happening in the screen and allows the user to hear the screen content before activating any control. It also enables the user to receive feedback and navigate the interface using an external keyboard or a Braille display. The navigation with VoiceOver is based on controlling the VoiceOver focus, a rectangle that marks which element is selected and can be activated at a given moment. When using VoiceOver, touching the screen will not directly activate any element - unlike during regular iOS operation. Instead, touching the screen becomes a means to explore the user interface without risk of undesirable item activation.

One of the main features of VoiceOver is the Rotor. The Rotor is a virtual dial that allows users to control the type of item they navigate by at a given moment (i.e. characters, words, headings or links), as well as to perform quick changes (i.e. change speech language, speaking rate or typing mode). Since the amount of touch screen gestures is limited, at least without stretching the levels of complexity and memorization, the existence of a Rotor enables not only several navigation options in different contexts but also changing between them (known as Rotor items). The Rotor can be customized and the Rotor item can be quickly switched at any time, ensuring a fast and fluid navigation.

VoiceOver comprises a wide range of settings and customizations. Considering the default settings, the main touch gestures to control its operation are the following [10]:

- Touch with one finger:

VoiceOver moves the VoiceOver cursor to the item touched and reports it to the user. It doesn't activate the selected item, as iOS usually does when VoiceOver is not active. The user can touch the screen in one point, move the finger continuously through the screen and VoiceOver will report in real time each element the user's finger passes by;

- Double tap with one finger:

  Activates the currently selected item selected by the VoiceOver cursor. It performs the same action as touching the screen during regular iOS operation, without Voiceover turned on;

- One finger flick left or right:

  Moves the VoiceOver cursor to the previous or next user interface item, respectively. The flick consists in briefly touching the screen with one finger and then move it quickly to the left or right;

- One finger flick up or down:

  To move focus to the next or previous element of the selected Rotor item, respectively;

- Two finger rotate clockwise:

  This gesture switches the Rotor to the next Rotor item, affecting the navigation performed by the one finger flick up or down gesture. This rotation gesture can be compared to increasing the volume in a virtual volume dial, or the equivalent of rotating an image to the right with VoiceOver turned off;

- Two finger rotate counter-clockwise:

  This gesture switches the Rotor to the previous Rotor item, affecting the navigation performed by the one finger flick up or down gesture. This rotation gesture can be compared to decreasing the volume in a virtual volume dial, or the equivalent of rotating an image to the left with VoiceOver turned off;

- Three finger flick up, down, left or right:

  To scroll to the next visible area of a page following natural direction. In other words, to scroll one page down, up, right or left, respectively. The flick consists in briefly touching the screen with three fingers and then move them quickly up, down, to the left or to the right;

- Two finger double tap:

  This gesture, also known as magic tap, allows the user to quickly stop or play audio or video, to accept incoming calls or to end calls in progress;

Some advanced gestures encompass:

- Split tap:

  Item activation can also be performed with what is called the split tap, that consists in touching an item with one finger and then, without lifting it, touch the item with another finger. This alternative activation gesture merges the item selection and activation by requiring less steps to activate an item. So, users can activate an item immediately after realizing they have located the desired element;

- Two finger scrub:

  This gesture is used to perform a properly configured back gesture, which is equivalent to the one achieved by dragging one finger from the left side of the screen during regular iOS operation. A VoiceOver user can perform it with a two finger scrub, i.e. touching the screen with two fingers and drawing a letter "Z";

- One finger double tap and hold:

  The user is given the option to temporarily bypass VoiceOver gestures. After performing a one finger double tap and hold, the device will register the continuation of this gesture as if VoiceOver was not turned on (i.e. touch a slider, perform the double tap and hold and then move the finger right to increase the slider value);

- Two finger flick up or down:

  VoiceOver reads continuously all the interface's content from the beginning or from the current focused element, respectively. This command is usually performed to read a web page or document, so the user does not need to constantly flick right to advance on the content. Also referred to as Say All, the continuous reading is interrupted when the user touches an element or performs another gesture.

Depending on the iOS device (namely on the existence of a Home button), there are also specific gestures to open the notification centre and the control center. In addition, VoiceOver offers several typing modes, like direct touch typing that allows the use of the keyboard as if VoiceOver was turned off (for faster typing), options to input text by handwriting or to type simulating an on-screen Braille keyboard. VoiceOver also allows the user to turn on the screen curtain (with a three finger triple tap), turning the screen black for increased privacy (e.g. typing passwords, reading private messages).

3.2 AppleVis

To the best of our knowledge, the only repository of accessibility data of iOS applications can be found at AppleVis.com. Its publicly available iOS bug and app repositories were never explored in an academic context and they are an excellent source of information to analyze the state of vision accessibility in the iOS ecosystem.

AppleVis is a website that presents itself as the leading online resource for blind and low vision users of Apple devices [14]. Although it is community based and independent from Apple, AppleVis is a recommended resource on Apple's official page dedicated to vision accessibility [7].

Counting on a large and active worldwide user base, AppleVis grounds its contribution on the combination of the knowledge and experience of the individuals in the community with the goal of leading every member to greater fulfillment and independence, by taking advantage of Apple devices' accessibility features [14].

From getting started guides to product reviews, feature requests, idea discussions and advanced tutorials, AppleVis presents a variety of resources to the blind and low vision community of Apple users.

No statistics are provided about the percentage of low vision and fully blind users who are registered on AppleVis. However, due to the focus on Voiceover Performance, we extrapolate that most of the users are fully blind and rely on VoiceOver to use their Apple devices.

Finally, AppleVis also makes available a Bug Tracker and an App Directory. The Bug Tracker is a registry of every active and fixed vision accessibility related bug in iOS and macOS, focusing on both the operating system and native apps [13]. It is maintained and updated by the AppleVis editorial team. The App Directory, with repositories for every Apple operating system (iOS/iPadOs, macOS, watchOS and tvOS), presents a list of user submitted app reviews focusing on the app's accessibility and usability from the perspective of a user with vision disabilities [12].

Since the focus of this investigation is on iOS, analyzing the iOS Bug Tracker and App Directory has been of utmost importance for the consecution of the objectives defined.

3.2.1 Data Collection

Given the relevance of the data present on the AppleVis Bug Tracker and App Directory for the purpose of this work, the website administration was requested such data. Once this information could not be provided in any aggregated format besides the html content of the web pages themselves, a web scraping project was undertaken to gather all the data needed for a proper analysis in an efficient and automated way.

Although the data is public and anyone can access it, AppleVis was asked permission for the automation of the web scraping process to collect the data and its subsequent use in this investigation. The permission was granted by the website administrators, under the compromise of not exploring the personal information of the AppleVis' users.

The web scraping project was developed using Python and Beautiful Soup. The content of each bug report and app review, except user information and forum comments, were downloaded, aggregated and converted to a JSON list of unstructured data, comprising only the title and relevant html content of the page.

After downloading the information, a small program was built to treat the data and convert the raw html content into properly defined objects, only containing the relevant data for the analysis. Upon treatment, the data was subsequently parsed to a duly formatted CSV file in order to be imported to other tools and analyzed, as either a spreadsheet or a panda data frame.

The content of the downloaded iOS Bug Tracker and App Directory can be found in Appendix A, in the respective sheets. The only addition to this data has been the automatically generated ID column.

In the following two chapters, the Bug Tracker and App Directory, respectively, will be further scrutinized and explored in the context of this work's objective.

Chapter 4

Proposed Classification

The process followed in this investigation involved two different stages: firstly, the construction of the classification of iOS vision accessibility errors; and secondly, the validation of such created classification.

To construct the classification of iOS accessibility errors, the following steps were performed:

- Analysis of the information in the iOS Bug Tracker;
- Extension of the Bug Tracker's data by grouping bugs into similar problems in order to understand their scope;
- Mapping the identified problems from the bugs with the existing accessibility guidelines to create distinct types of errors;
- Exploration of the defined types of errors in finer detail to identify specific error subtypes;
- Inspection of the extended information to a further understanding of iOS bugs.

In turn, the steps taken to validate the created classification were the following:

- Analysis of the information in the iOS App Directory;
- Classification of a subset of the reported accessibility problems according to the proposed classification, creating new types or subtypes if needed;
- Aggregation of the information in the iOS App Directory and analysis of the classification's comprehensiveness;
- Development of a questionnaire to validate if the classification is intuitive for screen reader users;
- Analysis of the questionnaire results.

4.1 Subject - iOS Bug Tracker

The iOS Bug Tracker is a registry of every active and fixed vision accessibility related bug present in iOS since iOS 8, focusing on both the operating system and native apps.

The bug reports, submitted by the AppleVis editorial team, concern single bugs and contain detailed steps to reproduce them and possible workarounds. Fixed bugs are marked as inactive, usually upon the release of a new iOS version, and they are accompanied by a post detailing the new and fixed accessibility bugs in each iOS version. Users may consult this information to decide whether they should update to a new iOS version or wait for the following one - in case some important feature is not working from an accessibility perspective.

On January 29, 2021, the iOS Bug Tracker consisted of 249 bug reports, of which 23 were active in iOS 14.4 and 226 had already been fixed in the current or previous iOS versions (AppleVis: Bug Tracker, 2021). The earliest bugs in the repository date from iOS 8.0, released in 2014 [13].

From the reports aggregated by the Bug Tracker, several information can be consulted about each reported bug:

- Title – A summary of the bug;
- Description – A more detailed explanation of the bug;
- Steps to Reproduce – Details about how to reproduce the encountered bug;
- Severity – The degree of severity of the encountered bug, regarding impact in the user experience, which can only assume the values Serious, Moderate and Minor;
- Category – AppleVis categorization of bugs (discussed below);
- First Encountered – The first iOS version in which the bug was found;
- Devices Encountered – The device in which the user found and tested the bug;
- How Often – The frequency in which the bug occurs, assuming only the values Always, Sometimes and Infrequently;
- Workaround – When existent, how to avoid the bug;
- Apple Feedback # – The identifier of the bug when reported to Apple;
- Status – If the bug is active or fixed;
- Fixed In – When the bug's status is fixed, in which iOS version the fix occurred.

In the AppleVis Bug Tracker, the Category attribute can assume the following values:

- Low Vision:

  Bugs related with visual problems that affect users with low vision;

- VoiceOver Announcements/Feedback:

  Bugs that affect VoiceOver users and cause erroneous or lack of VoiceOver output in a certain situation;

- VoiceOver Speech/Voices:

  Problems that affect the VoiceOver voices, in terms of pronunciation and content being read inaccurately;

- Braille:

  Problems that affect VoiceOver users that rely on Braille input (e.g. external Braille keyboards and on-screen Braille input) and/or output (e.g. external Braille displays);

- Mail:

  Bugs related with iOS's native Mail app;

- Messages:

  Bugs related with iOS's native Messages app;

- Phone:

  Bugs related with iOS's native Phone app and with receiving and making phone calls;

- Safari:

  Bugs related with iOS's native Safari web browser;

- Bluetooth:

  Bugs related with BlueTooth devices (e.g. BlueTooth speakers);

- Miscellaneous:

  Bugs of other nature that do not belong to the previous categories.

4.2 Bug Tracker Extension

As explained by Clegg-Vinell et al. [19], relating problems to the existing guidelines is a challenging task that requires deep knowledge on those guidelines, on the supporting documentation and also, prior experience. After researching the guidelines, verifying the information available in the iOS Bug Tracker and dissecting the title, category, description and steps to reproduce of each bug, the utter complexity of establishing a direct link between each bug report and the guidelines becomes evident.

In her study, Alajarmeh [3], after collecting a number of problems from users, combined and aggregated them into a reduced list summarizing the major problems found. Although Bug Tracker displays a list of categories, they are not particularly informative about the major problem presented in each bug.

Nevertheless, the complexities of understanding the major problem each bug represents may be mitigated by the additional information one can extrapolate from the original data.

Analyzing the categories available in the iOS Bug Tracker revealed that they can be grouped into two distinct main areas:

- Bugs Related with Specific Apps – The Mail, Messages, Safari and, to some extent, Phone categories are related with specific iOS native apps. When explored thoroughly, more specific app related bugs can be found on this bug tracking list, such as bugs related to the Music, Weather or Calendar apps, amongst others;
- Global Operating System Bugs – Including the Braille, Low Vision, BlueTooth and Voiceover specific bug categories, which can be felt throughout the whole iOS experience.

These bugs affect vision impaired users in different ways. While bugs related to the whole operating system are felt globally when using the device, regardless of the app, bugs specific to certain apps and contexts concern a single app. The former can only be solved by fixing the operating system, but the latter only affects the user interface elements of a certain app. Thus, by concerning an isolated app, the bugs present in the iOS Bug Tracker offer valuable insights to interpret the bugs reported in the iOS App Directory, as a directory that deals with apps developed by independent developers that use specific iOS APIs and are not able to change the operating system.

Considering their major and important dissimilarity, an App Related attribute was constructed. It consists in a boolean value classifying a bug as related to a specific app (True) or to the iOS usability as a whole, therefore felt in all the system's apps (False).

A System Function attribute was also created, in order to group bugs by the app or context in which they are encountered, such as the aforementioned Mail, Messages and Safari categories, but also by other apps, (e.g. Music or Weather), and by system functionalities, such as notifications, Home screen, Lock screen, Siri, amongst others. This attribute has been crucial to group the bugs by app and clearly understand the problems within their context.

Another singularly informative attribute was extracted from the data available in the iOS Bug Tracker. For bugs already fixed, with the Fixed In attribute present, we were able to calculate how many iOS versions were needed in order to fix the bug. This attribute is called Gap to Fix and it expresses how many major and minor iOS versions were needed to fix a bug. We considered that the unit would represent major versions whilst the decimal would reflect minor ones. For instance, a bug first encountered in iOS 9.0 and fixed in iOS 9.3 represents a gap to fix value of 0.3, whereas

a bug first encountered on iOS 12.0 and fixed in iOS 13.4 has a gap to fix of 1.4. Thus, on the latter example, one major version and 4 minor ones were needed to fix the bug. This value correlates positively with the time a bug took to be fixed.

To simplify the data analysis process, a boolean Is Active column was also generated, classifying the active bugs as True and the fixed ones as False. The closed textual attributes Severity and How Often were also converted to numeric scales, in which an increase represents a less accessible situation (e.g.: Severity - 1: Minor, 2: Moderate and 3: Serious; and How often - 1: Infrequently, 2: Sometimes and 3: Always).

After creating the App Related and System Function attributes (Appendix B), all bug reports were analyzed and classified according to the their accessibility problem. Identifying the accessibility problems described in the bug reports was accomplished by reading the bug's title, description and steps to reproduce, actively searching for common aspects or patterns that somehow associated some of them.

Re-categorizing such problems was an iterative process. A first iteration was accomplished ordering the bugs by category. Another iteration grouped them by System Function and then, three more iterations were performed, ordering the list by the problems previously identified. Once the list comprised more than 200 elements (249 in the last iteration, since new iOS versions were released during the corse of this investigation and new bugs were added), each iteration was performed in no more than three days and with a five day break between iterations, to allow a more distanced approach in each.

The final list of problems comprised the following 16 elements [1]:

I Announcements – When VoiceOver either talks but is not supposed to, does not talk but is supposed to or provides incorrect information;

II Audio – Bugs related to incorrect sound output;

III Bluetooth – Bugs related to output to BlueTooth devices;

IV Braille – Problems related to navigation or reading using Braille displays;

V Braille Typing – Problems related to typing Braille, with external Braille displays or the Braille Screen Input functionality;

VI Color Accommodations – Bugs related to the OS color accessibility features (Invert Colors and Dark Mode);

VII Customizing – Bugs referring to the iOS Home screen customization and widgets;

VIII Keyboard Navigation – Bugs related to navigation with an external keyboard;

IX Labeling – Bugs related to unlabeled or incorrectly labeled elements;

[1] As presented in Appendix B

X Low Vision – Bugs related to low vision functionalities that affect the expected size of the content;

XI Low Vision/Responsiveness – Bugs related to low vision functionalities that make the system unresponsive;

XII Navigation – Bugs that affect the structured navigation through the interface or make actions inexecutable;

XIII Responsiveness – Bugs that make the system lag or crash;

XIV Typing – Bugs related to entering text in the system;

XV VoiceOver Sounds – When specific sounds (that represent common actions) are not outputted or are wrongly outputted by VoiceOver;

XVI Voices – Bugs specifically related to VoiceOver voices configuration, performance and pronunciation.

Finally, these identified problems were mapped with the guidelines present in the standards previously exposed, namely with the principles from the WCAG mobile guidelines [41] and the high-level topics from the BBC Mobile Accessibility Guidelines [16]. For the purpose of this investigation, instead of only focusing on one of them, both sets of guidelines were included, due to WCAG's completeness and BBC's intuitiveness.

The process to accomplish this mapping involved analyzing each of the aforementioned problems and verifying if there were any WCAG principle and BBC high-level topic that mentioned them. The WCAG principle and BBC high-level topic in which the match could be found were registered. Then, every bug description the problem encompassed and respective steps to reproduce were dissected to test if they matched any of the the recommendations that principle and topic comprehend. If they did, the mapping was confirmed and we proceeded to the next problem. Otherwise, we searched for other principle or topic that covered that bug's information.

The mapping (Appendix B) was also an iterative process. The first iteration was accomplished ordering the bugs by problem and the second one ordering the list by such mapping. Each iteration was performed in no more than three days and with a five day break between iterations. After the process, some bugs initially classified with the same problem turned out to map to guidelines from different WCAG principles or BBC topics. This was particularly noticeable on bugs pertaining to the Navigation problem, once it incorporated a broad range of issues, but also on Typing, Responsiveness and Labeling. Nonetheless, most of the problems were narrower and mapped entirely to the same guideline group.

Hence, a new attribute was created and named Type. The naming of this new feature's values was mostly based on the guidelines - despite having also been influenced twice by Bug Tracker categories and once by an identified problem.

4.3 Classification

After the mapping with the guidelines, the following bug types were established:

I Alternative Interaction – Bugs related to accessible alternative interaction modes, such as alternatives to complex gestures, keyboard navigation and Braille display navigation, output and input. Mapped from WCAG's principle Operable and partially from BBC's Design;

II Contextual Notifications – Bugs related to voice, audio effects or vibration feedback that Voiceover provides to replace visual cues. Mapped from WCAG's Robust and from BBC's Notifications and Audio and Video;

III Focus and Structure – Bugs related to VoiceOver focus' behavior and the user interface's natural interaction flow. Mapped from WCAG's principle Understandable and BBC's Focus, Links, Scripts and Dynamic Content, and Structure;

IV Low Vision – Bugs related to visual aspects of the interface that affect low vision users. Mapped from WCAG's Perceivable and BBC's Design;

V Responsiveness – Bugs that lead VoiceOver or the system to become unresponsive or crash. There was no guideline covering this type of bugs;

VI Speech Output – Specific bugs related to the available VoiceOver text to speech (voice) engines, such as pronunciation, misread words or delays. There was no guideline covering this type of bugs;

VII Text Equivalents – Bugs related to providing alternative textual description of non-textual elements, such as buttons, images, graphics or inaccessible text. Mapped from WCAG's principle Perceivable and BBC's Editorial, Images, and Text Equivalents;

VIII User Input – Bugs related to the user input of text or other information. Mapped from WCAG's Robust and BBC's Forms.

The bug types covered all the WCAG principles and BBC topics. Despite not being extensive on a guideline to guideline basis, the mapping to all the high-level elements provides confidence about the coverage of our classification.

The Responsiveness type could not be mapped. Within the scope of this investigation, we were led to assume that the bugs this type encompass, theoretically, would not happen if the guidelines had been met. The Speech output was not mapped once it is a specific problem of the screen reader operation, and the guidelines are oriented to developing accessible software and its interaction with assistive technology, not to the operation of such assistive technology.

After a more thorough inspection of the bugs grouped by type and the guideline mapping process in its origin, as well as considering the previously obtained problems, this classification was further scrutinized into a more granular level of detail.

Thus, a new attribute, called Subtype, emerged. It identifies specificities in each bug type that sustain a clearer classification of accessibility errors. The names of the subtypes were based on WCAG and BBC guidelines, bug report categories, frequently reported issues and even on an accessibility property in SwiftUI's API [13]. However, some subtypes were conceived based on directly sub-grouping reported causes (e.g. Braille Typing, Wrong Label), exhaustively analyzing the problem and its context (e.g. Unreachable Elements, subtypes from Responsiveness) or even understanding the intent of the VoiceOver functionality affected by the bug (e.g. Automated Functionality, Fast Navigation), as detailed in Appendix B.

The subtypes, grouped by type and followed by a brief explanation and some bug exemples, are presented below:

I Alternative Interaction

- Automated Functionality – Operations that VoiceOver automates

  Examples:

  Audio destination incorrectly appears in the Rotor with Apple AirPods;

  The performance of VoiceOver's Recognition features is inconsistent and unreliable.

- Braille – Reading and navigating using a braille display

  Examples:

  The Detected Text feature that is part of the VoiceOver Recognition features does not work with Braille displays;

  Extraneous feedback given to Braille users when navigating the Mail app;

  Problems pairing and re-pairing the Freedom Scientific Focus line of Braille displays.

- Braille Typing – Inputing text using a Braille display or the Braille Screen Input

  Examples:

  Braille Screen Input cannot be used to type text on webpages;

  Text input with a Braille display is unreliable when typing quickly;

  While typing using a braille display, if a notification or other flash message comes in, the cursor will jump to another random point in the text field.

- Fast Navigation – VoiceOver Rotor options to jump between dynamic interface elements

  Examples:

  You cannot use the VoiceOver Rotor to navigate by line when composing an email in the Mail app;

  The Misspelled Words tool is available from the VoiceOver Rotor in Safari text fields, but does not work.

- Gesture Alternatives – VoiceOver Rotor actions that replace complex gestures

  Examples:

When re-ordering widgets on the Today View, occasionally, selecting the Drop Before option in the VoiceOver Rotor will create a widget stack instead;

When listening to an audio file in the Files app, flicking up or down on the playback scrubber control does not rewind or fast forward playback;

The VoiceOver Rotor no longer consistently switches to the Actions Rotor item when VoiceOver focus lands on an element which has actions available;

The Mark Read/Unread Flags do not stick in the Mail app if set with Rotor actions;

The Scrub gesture does not work in Settings > iCloud > Storage > Buy More Storage.

- Keyboard Navigation – Navigation using an external keyboard

  Examples:

  On occasion, pressing VO+Space on an external keyboard will trigger the Context Menu rather than activate the item VoiceOver focus is on;

  Unable to access the Status Bar with the VoiceOver keyboard shortcut.

II Contextual Notifications

- Announcements – Reporting an action that is performed or related to notifications

  Examples:

  VoiceOver fails to indicate Touch ID unavailability upon first restart of a device;

  VoiceOver may not always automatically announce the time when you press the Power button to wake your device;

  During a Say All, VoiceOver does not ignore incoming text messages.

- Sound Hints – Sounds that replace visual indicators

  Examples:

  The audible tone used by Safari to indicate that a webpage is loading may continue to play after the page has completed loading;

  Misleading sound and haptic feedback when deleting characters in some text fields;

  Missing audible tone after your device has been successfully unlocked with Face ID.

III Focus and Structure

- Extraneous Elements – When elements that are supposed to be hidden or covered appear in the interface while using Voiceover

  Examples:

  VoiceOver finds and reads elements on the Mini Player of the Music app, which are not visually present or actionable;

  Duplicated Heading at the top of the For You tab of the native News app.

- Focus – When the VoiceOver focus moves to an interface element without the user's intent

Examples:

VoiceOver focus will occasionally jump to the Status Bar when swiping the screen;

VoiceOver focus may move to the first UI element on the screen after you have interacted with another element on the screen;

Banner notifications may, on occasion, stay on screen even though banner style is set to temporary.

- Item Activation – When an item is not activated after the user performs the activation gesture

  Examples:

  On occasion, it is not possible to expand grouped notifications in Notification Center when VoiceOver is enabled;

  When editing a video in the Photos app, the Done button becomes unavailable after editing the start or end point of the video using the adjustable action in the VoiceOver Rotor.

- Navigation – When the page navigation controls do not work as expected

  Examples:

  In the native Music app, navigating by Heading is unreliable for VoiceOver users;

  Ability to scroll the screen up and down is missing on some occasions on the Today screen.

- Unreachable Elements – When Voiceover can not detect an element of the user interface

  Examples:

  Title button for group message conversations not detected by VoiceOver;

  On some devices VoiceOver users are not able to locate by touch the Status Bar after one app has opened a second app; Problems locating attached files when composing a new email in the Mail app.

IV Low Vision

- Color Accomodations – When the Invert Colors option does not work as expected

  Example:

  Smart Invert and Classic Invert may not behave reliably or consistently.

- Low Vision – Misrepresentation of visual elements

  Example:

  The VoiceOver cursor will occasionally disappear or become very large.

- Responsiveness – When the Low Vision commands do not perform the desired actions

  Examples:

Multitouch gestures are inconsistent and unreliable when both VoiceOver and Zoom are enabled;

You cannot enter Edit mode on the Home screen when both VoiceOver and Zoom are enabled.

V Responsiveness

- Audio – When audio output fails or is unresponsive

  Examples:

  Audio may, on occasion, breakup and crackle;

  Audio does not always return to correct level after being interrupted by VoiceOver when audio ducking is enabled.

- Context Crash – When an app or VoiceOver crash but the device operation continues

  Examples:

  The Settings app crashes when trying to add or edit email signatures;

  A two finger swipe down,"read all from current position", in the News app may cause VoiceOver to crash;

  Utilizing the Back button in Safari causes VoiceOver to become unusable.

- Context Inoperability – When an action is not performed after the user performs the needed steps

  Examples:

  Screen auto-locks while VoiceOver is reading text;

  Siri may not respond if you invoke it whilst VoiceOver focus is in a text field and Direct Touch Typing is selected;

  Issues using two finger double tap gesture to answer or end a phone call.

- Global Crash – When the whole system crashes and the device needs to be restarted

  Examples:

  Device may crash and respring when using the VoiceOver actions menu to close the last used app via the app switcher;

  VoiceOver may become unresponsive after ending a call in the Phone or FaceTime apps.

- Temporary Inoperability – When VoiceOver has a temporary delay and recovers after a short period of time

  Examples:

  Device freezes for a short period after exiting an app;

  VoiceOver may on occasions become unresponsive and need to be restarted.

VI Speech Output

- Voices – When the VoiceOver voices do not read as expected

  Examples:

  Prices preceded by the British pound symbol are not spoken correctly by VoiceOver;

  There are several pronunciation and inflection issues with the new Irish Siri Female voice when it is used with VoiceOver;

  Enhanced quality voices will occasionally revert to the default variant and require a re-download.

VII Text Equivalents

- Incomplete Information – When Voiceover reads some but not all the visually available information

  Examples:

  VoiceOver may not announce new item badges on Home screen icons;

  VoiceOver does not speak all of the information from the Playback Destination button when audio is routed to an external device;

  When viewing a list of articles in the native News app, VoiceOver does not always correctly indicate those which have been read.

- Label with Extraneous Text – When the element is correctly labeled but some undesired extra text is part of the label

  Examples:

  Various UI elements in Today View widgets have their VoiceOver label prepended with the extraneous text "today";

  Voiceover announces "new line" when announcing a language with multiple variants in the Translate application;

  On the native Weather widget, VoiceOver speaks the word "degrees" twice.

- Unclear Label – When the label is not clear about which action it triggers

  Examples:

  The button when adding a new event to the Calendar app needs a more concise label;

  The selection status of the Dark/Light Mode toggle in the Control Centre needs to be more accurately stated to VoiceOver users.

- Unlabeled Element – When an element is not labeled and Voiceover users can not know its intent

  Examples:

  Voiceover encounters an unlabelled element when navigating in the Listen screen of the Translate application;

  VoiceOver does not read all of the Screen Time data presented as charts in the Settings app;

The buttons for the two personalised music playlists new to Apple Music in iOS 10 are unlabelled;

When using the Measure app on an iPhone 12 Pro or 12 Pro Max to automatically measure a person's height, the result is not recognized or spoken by VoiceOver;

Contact labels not spoken in app switcher.

- Unspoken Selection – When Voiceover reports the element but not its selection status

  Examples:

  No feedback on current selection status when editing the list of mailboxes/folders to be displayed in the native Mail app;

  VoiceOver does not announce the current selection status of the Love and Dislike buttons in the Music app.

- Wrong Label – When the available label does not represent truthfully the type, status or action of the item

  Examples:

  VoiceOver wrongly announcing the contents of the App Store search field after it has been cleared;

  Incorrect labels for "Clear Section" buttons in Notification Center;

  VoiceOver will on occasions wrongly indicate that some blocks of text are editable text fields.

VIII User Input

- Typing – Problems related to the input of text or other information

  Examples:

  On occasions, you may not be able to locate by touch the Space key on the on screen keyboard;

  Can not edit the name of lists in the Reminders app;

  VoiceOver not recognizing the input of text in some text fields on web pages.

By grouping the existent bug information to these types and subtypes, each bug type and subtype's frequency, severity and gap to fix can also be inferred.

4.4 Data Analysis

After the performed extension of the iOS Bug Tracker, we proceeded to the analysis of the most common and impactful bugs visually impaired iOS users face.

According to the App Related attribute, among the 249 bugs considered, 141 (57%) occurred globally throughout the operating system, whereas 108 (43%) arose solely in the context of an app.

Table 4.1 presents the distinct system functions [2] classified in the Bug Tracker extension, their occurrences and the extent of their impact.

The most frequent operating system functionality affected by vision accessibility bugs has been the Home screen, comprising 29 bugs, followed by VoiceOver commands (20), problems with Text fields (18) and with the Lock screen (12).

Regarding native iOS apps, the Mail app (20), Settings (13) and Music app (13), as well as Messages app (10) are the most affected by accessibility problems. Although most of its bugs occur locally, Settings bugs also impact the overall usage of the operating system, besides the highlighted bug related to VoiceOver voices.

Considering the two subsets of local app and global operating system bugs, we conclude that almost every system function is exclusively related to one of them. This clearly defined boundary is only trespassed by 3 of the identified system functions, specifically the aforementioned Settings but also Lock screen, containing one bug related to interacting with notifications from Messages, and Phone Calls, with two bugs that relate to the Phone app itself.

Table 4.2 shows, for each iOS version, how many bugs have been disclosed, how many were fixed and their severity, frequency and longevity (Gap to Fix).

iOS 13 is the version with more vision accessibility bugs reported, either on the operating system or native apps, comprehending 58 new bugs. Following it, iOS 8, 14 and 11 were the versions with more new bugs. Elseways, iOS 12 and 14 only introduced 14 and 23 new ones, respectively.

Regarding the severity of those bugs, a slight decrease can be discerned in iOS since iOS 12, with the bugs being classified between Minor and Moderate.

The frequency of the bugs has no recognizable pattern in their evolution, with the bugs being, on average, registered as Always regarding how often they occur.

The bugs' longevity, measured by the Gap to Fix attribute, has been slowly decreasing since iOS 8 - although iOS 12 presents itself as an outlier, with the bugs encountered in this version taking more iOS updates to be fixed. Nonetheless, from iOS 11 to iOS 13, the mean gap to fix has decreased almost 2 iOS minor versions. As the latest and currently available, iOS 14 is the version with less fixed bugs - the only ones considered to compute the Gap to Fix attribute. Thus, we opted to exclude it from this specific analysis.

In what concerns iOS bugs fixed in each version, besides iOS 8, each iOS version fixes roughly the same bugs as it introduces. Regarding the bugs' severity and frequency, no recognizable patterns of evolution were found. On average, bugs were classified between Minor and Moderate in their severity and as Always regarding how often they occur.

iOS 12 stands out as the version that solved the more severe, frequent and long standing problems. Moreover, being also the version with fewer bugs introduced, it may be regarded as the best iOS version for visually impaired users thus far.

[2] Each of these system functions is more detailed in Appendix B

Table 4.1: Bug Tracker Extension – System Functions and App Related

System Function	**Total**	**App Related**
App Store	4	Local
Audio	8	Global
Braille	4	Global
Calendar app	1	Local
Control Center	2	Global
Files app	2	Local
Find my app	2	Local
Global	11	Global
Home app	2	Local
Home screen	29	Global
Lock screen	12	Global (and 1 related to Messages app)
Mail app	20	Local
Mail/Notes apps	2	Local
Measure app	1	Local
Messages app	10	Local
Music app	13	Local
News app	8	Local
Notes app	2	Local
Notifications	8	Global
PDF	1	Global
Phone app	4	Local
Phone Calls	7	Global (and 2 related to Phone app)
Photos app	3	Local
Podcasts app	1	Local
Reading	8	Global
Real-Time Text	1	Global
Reminders app	2	Local
Safari browser	7	Local
Settings	13	Local (and 1 related to VoiceOver voices with global impact)
Shortcuts	1	Local
Siri	2	Global
Status bar	6	Global
TV app	1	Local
Text fields	18	Global
Translate app	3	Local
VoiceOver commands	20	Global
Watch app	1	Local
Weather app	3	Local
Web content	5	Global
iPad specific	1	Global

Table 4.2: Encountered and Fixed Bugs per iOS Version

	iOS 8	**iOS 9**	**iOS 10**	**iOS 11**	**iOS 12**	**iOS 13**	**iOS 14**
Encountered Bugs	44	26	33	36	14	58	38
Severity	1.66	1.65	1.61	1.69	1.57	1.50	1.47
Frequency	2.64	2.46	2.42	2.58	2.71	2.48	2.61
Gap to Fix	0.60	0.55	0.54	0.53	0.64	0.34	0.18
Fixed Bugs	28	32	29	37	13	51	35
Severity	1.68	1.59	1.72	1.59	1.85	1.57	1.40
Frequency	2.75	2.41	2.59	2.38	2.85	2.57	2.51
Gap to Fix	0.30	0.54	0.51	0.44	0.73	0.50	0.43

*Severity and Frequency range between 1 and 3.

Furthermore, given the longevity of the fixed bugs per version, one can extrapolate that near half the bugs are only fixed in the following major iOS version. In fact, every version fixes as many bugs as it introduces (except iOS 8) and there are still active bugs.

Table 4.3 scrutinizes the overall distribution of the problems identified, as well as their occurrence at a global operating system level or in isolated apps.

The Navigation problem was the most frequently found in this repository of iOS bugs, categorizing one third of the total bugs, with 84 occurrences. The Labeling problem represents 18% of the bugs, with 46 occurrences, while Responsiveness bugs (21) are the third most representative problem (8%). The following 3 problems, Braille Typing (16), Braille (14) and Typing (14) can be grouped in two different ways. If considered as a whole, all Braille problems represent 12% of the bugs, whereas all Typing problems represent 11% of all the bugs. Regardless, any of those groupings would be the third most prevalent problem.

Concerning only the problems verified at the operating system level (Global), Navigation is also the most represented, registering 37 bugs (26%). It is followed by 15 Responsiveness bugs (10%) and 13 Labelling and Braille bugs (9% each). As aforementioned, grouping all Braille bugs would represent 17% of the operating system bugs - which would turn them into the second problem users face the most across iOS.

Scrutinizing native apps only, the problem Navigation accounts for almost half the bugs reported (43%), registering 47 bugs. Labeling bugs represent almost a third of the total (30%), with 33 instances. The third most prevalent problem at the app level, considerably less incident than the previous, is Typing, with 8 instances (7%). Discernibly, regarding the native iOS apps, the bugs are concentrated on Labeling and Navigation, jointly responsible for slightly less than three quarters (73%) of all bugs registered locally on the system apps.

As stated in Section 4.2, these two problems are very broad and encompass a variety of bugs. Nevertheless, with the creation of the Type and Subtype features, they became increasingly more detailed and dissectible. A complete analysis of the aggregated longevity, severity and frequency is only applicable after the segregation of these problems.

Table 4.3: Identified Problems

Problem	**Total**	**%**	**Global**	**%**	**Local**	**%**
Announcements	12	5.06%	9	6.82%	3	2.86%
Audio	3	1.27%	3	2.27%	0	0.00%
Bluetooth	1	0.42%	1	0.76%	0	0.00%
Braille	14	5.91%	13	9.85%	1	0.95%
Braille Typing	16	6.75%	11	8.33%	5	4.76%
Color Accommodations	2	0.84%	2	1.52%	0	0.00%
Customizing	8	3.38%	7	5.30%	1	0.95%
Keyboard Navigation	5	2.11%	4	3.03%	1	0.95%
Labeling	46	19.41%	13	9.85%	33	31.43%
Low Vision	1	0.42%	1	0.76%	0	0.00%
Low Vision/Responsiveness	2	0.84%	2	1.52%	0	0.00%
Navigation	84	35.44%	37	28.03%	47	44.76%
Responsiveness	21	8.86%	15	11.36%	6	5.71%
Typing	14	5.91%	6	4.55%	8	7.62%
VoiceOver Sounds	8	3.38%	5	3.79%	3	2.86%
Voices	12	5.06%	12	9.09%	0	0.00%

Table 4.4 displays the bugs' distribution by Type, those types' distribution by the global or local contexts as well as their severity, longevity and frequency.

Alternative Interaction is the most common bug type (31%), followed by Focus and Structure (20%) and Text Equivalents (18%). The bugs of these types differ in how they are distributed. While Alternative Interaction bugs are predominantly felt through the operating system, Text Equivalents bugs are mostly present in the iOS native apps. Only the Focus and Structure and the User Input types register the same trend as the latter, whereas the other types are more represented in the operating system. Particularly, Speech Output and Low Vision types register bugs only at the operating system level.

With the highest severity level, Low Vision bugs are the ones with more impact on users, followed by User Input and Responsiveness. Text Equivalents are, in turn, the least severe bug type.

Table 4.4: Classification – Bug Types

Type	**Total**	**%**	**Global**	**%**	**Local**	**%**	**Severity**	**Gap to Fix**	**Frequency**
Alternative Interaction	78	45.61%	53	60.23%	25	30.12%	1.74	0.55	2.65
Contextual Notifications	20	11.70%	14	15.91%	6	7.23%	1.40	0.56	2.30
Focus and Structure	52	30.41%	21	23.86%	31	37.35%	1.50	0.46	2.62
Low Vision	6	3.51%	6	6.82%	0	0.00%	2.17	0.66	2.00
Responsiveness	26	15.20%	19	21.59%	7	8.43%	1.92	0.57	2.04
Speech Output	12	7.02%	12	13.64%	0	0.00%	1.33	0.42	2.33
Text Equivalents	45	26.32%	12	13.64%	33	39.76%	1.20	0.27	2.82
User Input	10	5.85%	4	4.55%	6	7.23%	2.00	0.41	2.50

*Severity and Frequency range between 1 and 3.

Table 4.5: Classification – Subtypes

Code	Type	Subtype	Total	%	Global	%	Local	%	Severity	Gap to Fix	Frequency
1.1	Alternative Interaction	Automated Functionality	3	1.22%	3	2.17%	0	0.00%	1.33	1.10	2.67
1.2	Alternative Interaction	Braille	13	5.28%	12	8.70%	1	0.93%	2.08	0.45	3.00
1.3	Alternative Interaction	Braille Typing	17	6.91%	12	8.70%	5	4.63%	2.18	0.79	2.65
1.4	Alternative Interaction	Fast Navigation	7	2.85%	1	0.72%	6	5.56%	1.43	0.52	2.86
1.5	Alternative Interaction	Gesture Alternatives	33	13.41%	21	15.22%	12	11.11%	1.45	0.44	2.58
1.6	Alternative Interaction	Keyboard Navigation	5	2.03%	4	2.90%	1	0.93%	2.00	0.70	2.00
2.1	Contextual Notifications	Announcements	12	4.88%	9	6.52%	3	2.78%	1.42	0.55	2.25
2.2	Contextual Notifications	Sound Hints	8	3.25%	5	3.62%	3	2.78%	1.38	0.57	2.38
3.1	Focus and Structure	Extraneous Elements	6	2.44%	3	2.17%	3	2.78%	1.00	0.60	2.67
3.2	Focus and Structure	Focus	24	9.76%	9	6.52%	15	13.89%	1.54	0.54	2.46
3.3	Focus and Structure	Item Activation	4	1.63%	2	1.45%	2	1.85%	2.00	0.35	2.50
3.4	Focus and Structure	Navigation	9	3.66%	5	3.62%	4	3.70%	1.56	0.34	2.78
3.5	Focus and Structure	Unreachable Elements	9	3.66%	2	1.45%	7	6.48%	1.44	0.30	2.89
4.1	Low Vision	Color Accommodations	2	0.81%	2	1.45%	0	0.00%	2.50	1.00	2.00
4.2	Low Vision	Low Vision	1	0.41%	1	0.72%	0	0.00%	1.00	1.00	1.00
4.3	Low Vision	Responsiveness	3	1.22%	3	2.17%	0	0.00%	2.33	0.43	2.33
5.1	Responsiveness	Audio	5	2.03%	5	3.62%	0	0.00%	1.60	0.76	2.20
5.2	Responsiveness	Context Crash	7	2.85%	2	1.45%	5	4.63%	2.14	0.34	2.29
5.3	Responsiveness	Context Inoperability	5	2.03%	4	2.90%	1	0.93%	1.80	0.70	2.00
5.4	Responsiveness	Global Crash	3	1.22%	3	2.17%	0	0.00%	2.67	0.10	1.67
5.5	Responsiveness	Temporary Inoperability	6	2.44%	5	3.62%	1	0.93%	1.67	0.78	1.83
6.1	Speech Output	Voices	12	4.88%	12	8.70%	0	0.00%	1.33	0.42	2.33
7.1	Text Equivalents	Incomplete Information	9	3.66%	1	0.72%	8	7.41%	1.22	0.30	2.67
7.2	Text Equivalents	Label with Extraneous Text	6	2.44%	1	0.72%	5	4.63%	1.00	0.30	2.67
7.3	Text Equivalents	Unclear Label	3	1.22%	1	0.72%	2	1.85%	1.00	0.23	3.00
7.4	Text Equivalents	Unlabeled Element	18	7.32%	6	4.35%	12	11.11%	1.17	0.27	2.89
7.5	Text Equivalents	Unspoken Selection	3	1.22%	0	0.00%	3	2.78%	1.67	0.27	3.00
7.6	Text Equivalents	Wrong Label	6	2.44%	3	2.17%	3	2.78%	1.33	0.24	2.83
8.1	User Input	Typing	10	4.07%	4	2.90%	6	5.56%	2.00	0.41	2.50

*Severity and Frequency range between 1 and 3.

In what concerns longevity, Low Vision bugs are also the ones which fixing takes more iOS versions to be implemented, followed by Responsiveness, Contextual Notifications and Alternative Interaction. Logically, Text Equivalents is the type that takes less versions to be fixed, since these bugs are solvable by merely changing the elements' label.

As predictable, since the behavior of labels is usually immutable, Text Equivalents is also the bug type with highest frequency. Alternative Interaction and Focus and Structure also registered relatively high frequencies. Nonetheless, all bug types, on average, register a frequency equal or above Sometimes. The bugs that affect users least frequently belong to the Responsiveness and Low Vision types.

The analysis of the mean severity, longevity and frequency tends to even out the values in the types with more bug instances. The scrutiny of those values becomes clearer and more useful considering the bug subtypes.

Therefore, Table 4.5 presents the subtypes proposed within the scope of this investigation grouped by their respective types (as exhibited in Table 4.4), as well as the severity, longevity and frequency of each[3].

Gesture Alternatives is the subtype with more bug instances in the Bug Tracker, representing 13% of the registered bugs. Focus (9%) is the second most commonly found bug, whereas Braille typing (6%) and Braille (5%) are also abundant subtypes.

[3]The Code attribute comprises the arabic number that uniquely represents the type, corresponding to the roman number presented in Section 4.3, followed by a "." and the arabic number representing the alphabetic order of each subtype within its type.

Considering the subtypes experienced at the operating system level, Gesture Alternatives is the most common (15%), followed by Braille, Braille Typing and Voices (8.5% each).

As previously stated, this confirms the plentifulness of Alternative Interaction bugs on both the overall and operating system levels, as three of the four most common subtypes belong to this bug type. At the native app level, Focus bugs are the most common (14%), followed by Unlabeled Elements and Gesture Alternatives (both with 11%).

Not surprisingly, Global Crash bugs represent the most severe subtype, as they force the user to restart the device. Perhaps as a consequence of this severity, Global Crash bugs register the lowest longevity - being fixed approximately one minor iOS version after being encountered. Unexpectedly, according to the implied by Table 4.4, the Context Crash subtype is the only other subtype in the Responsiveness type that registers a severity near Serious, whereas the remaining Responsiveness subtypes' severities are below Moderate.

The subtypes from the Low Vision bug type are in opposite sides of the severity spectrum, as Color Accommodations and Responsiveness are, respectively, the second and third more severe bug subtypes, while the only Low Vision bug (code 4.2) is among the least severe. Although not so discrepantly, the subtypes from the Focus and Structure and Text Equivalent types also register dissimilar severities. In the former, Item Activation bugs exhibit a Moderate severity, Navigation, Focus and Unreachable Elements range from Moderate to Minor while Extraneous Elements register a Minor severity. Among Text Equivalents subtypes, Label with Extraneous Text and Unclear Label exhibit a Minor severity, while Incomplete Information, Unlabeled Element and Wrong Label register a slightly higher severity and Unspoken Selection shows a severity closer to Moderate.

The Alternative Interaction subtypes' severities are evenly distributed, near Moderate, while the Contextual Notifications subtypes are closer to the lowest value of the interval.

Besides the aforementioned Global Crash, the following bug subtypes that took less time to fix were all those belonging to the Text Equivalents type. The Context Crash subtype took 3 minor iOS versions to be fixed. Nonetheless, the remaining Responsiveness subtypes took longer to be solved, with at least 7 minor versions needed to these bugs' resolution.

Within the Focus and Structure type, the Unreachable Elements, Navigation and Item Activation subtypes registered 3 minor iOS versions of longevity, while Focus and Extraneous Elements bugs took longer (6 minor versions) to be fixed. This dissimilarity within a same type was also noticeable in the Low Vision and, specially, the Alternative Interaction types. Both types comprise subtypes that took near 1 major version to be fixed, while the other subtypes took almost 5 minor versions.

Regarding the frequency of the bugs, only Low Vision, Global Crash and Temporary Inoperability occur less than Sometimes. Indeed, all other bug subtypes were encountered more often by visually impaired users. Such fact supports the trend made evident when the classification's types were explored.

4.5 Discussion

By following the process described throughout this chapter, a classification of iOS vision accessibility bugs was developed. The process started with the data available at the AppleVis iOS Bug Tracker, that comprised bugs described by the AppleVis editorial team and shared with the community of visually impaired iOS users on the website [14].

From these data, the system functions in which the bugs were felt and the distinction between bugs felt in the operating system or only in the context of an app were extracted. The identification of common aspects between bugs led to the extrapolation of the problems experienced by users.

By mapping these problems with the accessibility guidelines from WCAG [41] and BBC [16], the Type attribute was created. Then, from the patterns discerned in the mapping process, the problems previously identified and common specificities common to some bugs, the Subtype feature was developed.

All these new attributes provided insights about the distribution, severity, longevity and frequency of iOS bugs, contributing to a deeper understanding of the state of vision accessibility on the platform.

Since the classification was created and named based on the user-centered information present in the AppleVis iOS Bug Tracker and classifies all bugs described, we can surmise that it typifies iOS accessibility bugs reported by visually impaired users.

Furthermore, as the WCAG [41] and BBC [16] guidelines were used to group the bugs into Types, we can also conclude that the proposed classification conforms to the available accessibility guidelines.

This proximity to the terminology employed by users to describe vision accessibility bugs and adherence to the guidelines can improve developers' understanding of bugs reported by users but also facilitate relating bugs with the existing guidelines.

The following section proceeds to the validation of this classification.

Chapter 5

Validation

To assess the classification's completeness and comprehensibility, the data available in AppleVis' iOS App Directory were employed. More specifically, an experimental process and a survey were applied aiming to answer the following research questions:

RQ1: Can the proposed classification effectively cover vision accessibility errors reported by users in iOS apps?

RQ2: Is the proposed classification perceived similarly by several visually impaired users?

5.1 Subject - iOS App Directory

The iOS App Directory presents a list of app reviews focused on their accessibility and usability, from a vision disabled users' perspective [12]. Any AppleVis user can submit this information, at any time, and regarding any app available on the iOS App Store. Thus, these reviews usually comprise an overall description of the app, all the accessibility problems faced by the reviewer as well as personal remarks relating to the experience with the app, the experience reporting eventual bugs to the app's developers, or both.

On January 8, 2021, the iOS App Directory comprised 2229 user submitted app reviews. Since then, AppleVis' editorial team has deleted some older or outdated reviews, reducing the number of available reviews. In order to include a broader data set, this investigation made use of the information downloaded on that date.

The iOS App Directory comprises the following information:

- Title – The name of the reviewed app;
- Last Modified – When the review was lastly modified;
- Category – The category of the app (e.g. Books, News, Weather);

- Version – The version of the app;
- Free or Paid – If the app is free, paid or free with in-app purchases;
- Device Tested On – The device in which the user tested the app;
- iOS Version – The iOS version in which the app was tested;
- Accessibility Comments – The user's comments regarding the app's accessibility;
- VoiceOver Performance – How VoiceOver performs in the app (values shown below);
- Button Labeling – How many buttons had clear labels (values listed below);
- Usability – The most relevant accessibility measure, describing how easy the app was to use (values detailed below);
- Other Comments – Diverse user comments on the app. Mostly, regarding the app's functionality, but also relevant accessibility related information.

In the iOS App Directory, the VoiceOver Performance attribute can only assume the following values:

– VoiceOver reads all page elements;
– VoiceOver reads most page elements;
– Not applicable for this app;
– VoiceOver reads a few page elements;
– VoiceOver reads no page elements.

Likewise, The Button Labeling attribute can solely assume the following values:

– All buttons are clearly labeled;
– Most buttons are clearly labeled;
– Not applicable for this app;
– Few buttons are clearly labeled;
– No buttons are clearly labeled.

In turn, the Usability attribute can exclusively assume the following values:

– The app is fully accessible without the use of VoiceOver;
– The app is fully accessible with VoiceOver and is easy to navigate and use;
– The app is fully accessible with VoiceOver, but the interface could be easier to navigate;
– There are some minor accessibility issues with this app, but they are easy to deal with;
– There are some accessibility issues with this app, but it can still be used if you are willing to tolerate these issues and learn how to work around them;
– The app is fully accessible with VoiceOver, but the interface makes the app very difficult to use;

– Some parts of the app are accessible with VoiceOver, but not enough to make it usable;

– The app is totally inaccessible.

5.2 Experimental Setup

To evaluate the classification's completeness and answer RQ1, a subset of the AppleVis' iOS App Directory app reviews was classified. The procedure involved analyzing the information available in the Accessibility Comments and Other Comments for each app review and finding the most adequate subtype classification to the problems the the reviewer reported. In the presence of reviews comprising more than one bug, multiple classifications were attributed to each review.

If a problem reported could not be mapped with any of the subtypes on the classification, A new category, able to express it, was created. On the other hand, if the information available was too short or vague to be mapped to any classification, or if an app had no accessibility problems reported, those two scenarios representing absence of relevant bug information were registered separately.

The iOS App Directory's subset of reviews was selected based on two criteria: the least accessible apps and the iOS version being equal or posterior to iOS 8.0, as it was the earliest iOS version in the bug reports from which the classification was attained.

The data preparation included the construction of numeric scales for the attributes with closed values, namely VoiceOver Performance, Button Labeling and Usability. In this regard, the more negative the values are, the less accessible the scenario is. Whereas zero represents the most neutral scenario and positive values represent increasingly more accessible scenarios.

For VoiceOver Performance and Button Labeling the new scale was a mere transposition of the aforementioned textual values to numeric values, ranging from -2 to 2, according to the levels of accessibility they expressed. However, for the Usability attribute, that relied on an even number of possible values (therefore, lacking a middlepoint) - contrarily to the advised for Likert scales [28] - the codification was more intricate despite having followed the same standards as the previous ones.

The final scales were as follows:

- VoiceOver Performance:

  2 – VoiceOver reads all page elements;

  1 – VoiceOver reads most page elements;

  0 – Not applicable for this app;

  -1 – VoiceOver reads a few page elements;

  -2 – VoiceOver reads no page elements.

- Button Labelling:

2 – All buttons are clearly labeled;

1 – Most buttons are clearly labeled;

0 – Not applicable for this app;

-1 – Few buttons are clearly labeled;

-2 – No buttons are clearly labeled.

- Usability:

  3 – The app is fully accessible without the use of VoiceOver;

  2 – The app is fully accessible with VoiceOver and is easy to navigate and use;

  1 – The app is fully accessible with VoiceOver, but the interface could be easier to navigate and use;

  0 – There are some minor accessibility issues with this app, but they are easy to deal with;

  -1 – There are some accessibility issues with this app, but it can still be used if you are willing to tolerate these issues and learn how to work around them;

  -2 – The app is fully accessible with VoiceOver, but the interface makes the app very difficult to use;

  -3 – Some parts of the app are accessible with VoiceOver, but not enough to make it usable;

  -4 – The app is totally inaccessible.

Besides the iOS version criteria, the apps were filtered by either having a non positive VoiceOver Performance or Button Labeling or an Usability evaluation ranging from totally inaccessible (-4) to recognizing that the interface could be improved (1).

The filtering criteria can be expressed as: iOS Version $\geq$ "iOS 8.0" and (VoiceOver Performance ≤ 0 or Button Labeling ≤ 0 or Usability ≤ 1).

After the application of these criteria, the assessment was performed on a subset of the 640 least accessible apps since iOS 8.0, representing 28.7% of the initial universe of 2229 app reviews. The analysis was manual, throughout 7 days in blocks of no more than 50 apps at a time.

The attribute Classification registered the ID of the type and subtype identified, as well as lack of information or new problems. When various subtypes or problems were found, they were registered separated by a "; ".

When the Classification contained several values, the analysis script created one instance of that review for each subtype classified. The variable Is Original registers if the review was from the initial classification (True) or if it has been processed by our script (False), for auditing reasons. The Type and Subtype attributes were automatically generated based on the classification.

The classification of the subset of app reviews and the list of all identified bug subtypes can be found in Appendix C.

5.3 Completeness Results

After aggregating the generated information, the following findings are revealed:

As portrayed by Appendix C, from the 640 reviewed apps, 82 (12.8%) had no accessibility problems reported, while 85 (13.28%) did not contain enough detail to extract any relevant information pertaining our classification. We assume the absence of accessibility problems in the app reviews may have been influenced by the criteria defined for this investigation. Notwithstanding, in the latter case, such incidence relates to lack of information provided in the review itself, such as "the interface is a little confusing", "the app used to work great but now it doesn't" or "some improvements are needed to make the app work better". In both cases, no information regarding the problem could be inferred.

On the other hand, from the remaining 473 app reviews, 793 different bug classifications were extracted. We found that 263 app reviews registered a single bug classification and the remaining 210 registered two or more classifications, yielding 530 bugs, approximately two thirds of the totality of bugs identified.

From the bug types summarized in the previous chapter, only the Low Vision category had no representation in the analyzed app reviews. Most of AppleVis' users being fully blind, in accordance with the priorly stated in Section 3.2), supports the conclusion that Low Vision problems are the least commonly found in these app reviews.

From the previously classified subtypes, and besides those relative to Low Vision bugs, few were not found in this analysis:

- Alternative Interaction: Braille – Braille is usually implemented on the screen reader level. Thus, when used in apps, the screen reader translates the interface to Braille from the same information it uses to output as speech. This substantiates that no specific problems related to Braille were found at the app level, as in the Bug Tracker only 1 out of 13 Braille bugs was registered in the scope of a native app;

- Responsiveness: Global Crash – No reviews have registered that the device crashed while using an app with VoiceOver. This confirms the information from the Bug Tracker, that only registered this subtype on a global operating system level.

Two new bug types were found during this process:

In 11 instances, users have reported problems with inaccessible maps. Classified as a subset of the Focus and Structure type, these bugs' behavior resembles Unreachable Elements', howbeit affecting the reading and interaction with maps. This was not part of the initial classification because Apple's native Maps app is accessible for VoiceOver users.

In 43 instances, the reviewers reported that an app was fully inaccessible with VoiceOver. This problem could not be grouped in any of the previously presented types, as it represents the total inaccessibility of an app to a VoiceOver user. Therefore, to insert this Completely Inaccessible subtype on, another type, called "Unusable", was added. This was not part of the initial classification because there was no instance of completely inaccessible apps on the iOS native apps.

Table 5.1: App Reviews by Type

Types	**Total**	**%**	**Local**	**%**	**App Reviews**	**%**
Alternative Interaction	78	31.33%	25	23.15%	38	4.79%
Contextual Notifications	20	8.03%	6	5.56%	15	1.89%
Focus and Structure	52	20.88%	31	28.70%	257	32.41%
Low Vision	6	2.41%	0	0.00%	0	0.00%
Responsiveness	26	10.44%	7	6.48%	32	4.04%
Speech Output	12	4.82%	0	0.00%	2	0.25%
Text Equivalents	45	18.07%	33	30.56%	385	48.55%
User Input	10	4.02%	6	5.56%	21	2.65%
Unusable	0	0.00%	0	0.00%	43	5.42%

5.4 Data Analysis

Table 5.1 presents a comparison of the types considering the total amount of bugs in the Bug Tracker (Total), the bugs related with iOS native apps in the Bug Tracker (Local) and the findings this investigation was able to obtain from the app reviews:

Almost half of the classified bugs from the app reviews were classified as Text Equivalents (48.55%), followed by almost a third of apps with Focus and Structure bugs (32.41%).

Although the latter corresponds to roughly the same percentage as in the iOS native apps (28.70%), the percentage of Text Equivalents bugs is considerably higher in developer apps than in Apple's own native apps (30.56%).

The third most common bug type was Unusable apps (5.42%), which attests that some developers still do not take accessibility into account.

The app reviews' percentages of Responsiveness, User Input and Contextual Notifications show a small decrease when comparing to their percentage in iOS native apps, and Low Vision and Speech Output are practically inexistent in both app sets. Contrarily, Alternative Interaction displays a drastic difference. Representing almost a quarter of the bugs in native apps (23.15%), its weight diminishes to less than 5% in the app reviews. Since we analyzed the theoretically least accessible apps, this may reflect a predominance of bugs in basic functionality and a shortage of more advanced accessibility functions that, if problematic, could have been represented by this type of bugs.

Table 5.2 outlines a comparison of the subtypes considering the total amount of bugs in the Bug Tracker (Total), the bugs related with iOS native apps in the Bug Tracker (Local) and the results extrapolated from the app reviews' analysis.

From this table, we conclude that the most common subtype of error is Unlabeled Elements, representing almost a quarter of all the classified errors in app reviews (23.96%). In the native apps, it was the second most found bug with half of this percentage (11.11%). Unclear Labels (13.87%) and Unreachable Elements (10.47%) are clearly the other most commonly found bugs in this evaluation. In fact, Unclear Labels registers a much higher percentage than the one verified in the native apps (1.85%). These three most frequent bugs represent 48% of all the bugs found in the app reviews.

Table 5.2: App Reviews by Subtype

Code	Type	Subtype	Total	%	Local	%	% Type	App Reviews	%	% Type
1.1	Alternative Interaction	Automated Functionality	3	1.20%	0	0.00%	0.00%	3	0.38%	7.89%
1.2	Alternative Interaction	Braille	13	5.22%	1	0.93%	4.00%	0	0.00%	0.00%
1.3	Alternative Interaction	Braille Typing	17	6.83%	5	4.63%	20.00%	4	0.50%	10.53%
1.4	Alternative Interaction	Fast Navigation	7	2.81%	6	5.56%	24.00%	2	0.25%	5.26%
1.5	Alternative Interaction	Gesture Alternatives	33	13.25%	12	11.11%	48.00%	28	3.53%	73.68%
1.6	Alternative Interaction	Keyboard Navigation	5	2.01%	1	0.93%	4.00%	1	0.13%	2.63%
2.1	Contextual Notifications	Announcements	12	4.82%	3	2.78%	50.00%	12	1.51%	80.00%
2.2	Contextual Notifications	Sound Hints	8	3.21%	3	2.78%	50.00%	3	0.38%	20.00%
3.1	Focus and Structure	Extraneous Elements	6	2.41%	3	2.78%	9.68%	39	4.92%	15.18%
3.2	Focus and Structure	Focus	24	9.64%	15	13.89%	48.39%	34	4.29%	13.23%
3.3	Focus and Structure	Item Activation	4	1.61%	2	1.85%	6.45%	42	5.30%	16.34%
3.4	Focus and Structure	Navigation	9	3.61%	4	3.70%	12.90%	48	6.05%	18.68%
3.5	Focus and Structure	Unreachable Elements	9	3.61%	7	6.48%	22.58%	83	10.47%	32.30%
NA	Focus and Structure	Inaccessible Maps	0	0.00%	0	0.00%	0.00%	11	1.39%	4.28%
4.1	Low Vision	Color Accomodations	2	0.80%	0	0.00%	NA	0	0.00%	NA
4.2	Low Vision	Low Vision	1	0.40%	0	0.00%	NA	0	0.00%	NA
4.3	Low Vision	Responsiveness	3	1.20%	0	0.00%	NA	0	0.00%	NA
5.1	Responsiveness	Audio	5	2.01%	0	0.00%	0.00%	8	1.01%	25.00%
5.2	Responsiveness	Context Crash	7	2.81%	5	4.63%	71.43%	6	0.76%	18.75%
5.3	Responsiveness	Context Inoperability	5	2.01%	1	0.93%	14.29%	6	0.76%	18.75%
5.4	Responsiveness	Global Crash	3	1.20%	0	0.00%	0.00%	0	0.00%	0.00%
5.5	Responsiveness	Temporary Inoperability	6	2.41%	1	0.93%	14.29%	12	1.51%	37.50%
6.1	Speech Output	Voices	12	4.82%	0	0.00%	NA	2	0.25%	100.00%
7.1	Text Equivalents	Incomplete Information	9	3.61%	8	7.41%	24.24%	29	3.66%	7.53%
7.2	Text Equivalents	Label with Extraneous Text	6	2.41%	5	4.63%	15.15%	10	1.26%	2.60%
7.3	Text Equivalents	Unclear Label	3	1.20%	2	1.85%	6.06%	110	13.87%	28.57%
7.4	Text Equivalents	Unlabeled Element	18	7.23%	12	11.11%	36.36%	190	23.96%	49.35%
7.5	Text Equivalents	Unspoken Selection	3	1.20%	3	2.78%	9.09%	28	3.53%	7.27%
7.6	Text Equivalents	Wrong Label	6	2.41%	3	2.78%	9.09%	18	2.27%	4.68%
8.1	User Input	Typing	10	4.02%	6	5.56%	100.00%	21	2.65%	100.00%
NA	Unusable	Completely Inaccessible	0	0.00%	0	0.00%	NA	43	5.42%	100.00%

In the 10 most frequent found bugs in the app reviews, only the Completely Inaccessible apps (5th) were not part of the Text Equivalents or Focus and Structure types.

Immediately after the top 10, Gesture Alternatives bugs are the first entry not pertaining to the two previously mentioned most frequent bug types, with less than a third (3.53%) of the percentage it had on the native apps (11.11%), in which it was the third most common bug. The Focus subtype was the most frequent bug in native apps (13.89%) and, although on the top 10, it is the least frequent within the Focus and Structure type bugs found in the app reviews.

Excluding the aforesaid three most common bugs, the following 10 most frequent bugs comprise 41.62% of the total, each representing more than 2%. From these latter (7 of which are part of the previously mentioned top 10), only Completely Inaccessible, Gesture Alternatives and Typing were not part of the two aforementioned most frequent bug types - Text Equivalents and Focus and Structure.

5.5 Intuitiveness

To test the intuitiveness of the classification and answer RQ3, we created a questionnaire to be shared online amongst screen reader users.

Although this investigation and specifically RQ2 mention visually impaired users, the predominance of VoiceOver bugs in the AppleVis database implies screen reader knowledge to respond to this questionnaire (as detailed below), thus excluding low vision users who do not use a screen

reader. Despite the reduction in the pole of potential visually impaired respondents, the questionnaire can reach a broader set of users by not imposing that the respondents have to be visually impaired, as its target can include people who have knowledge about using a screen reader for professional reasons.

Given the substantial size of the classification, specially considering its sub types, we opted to exclude the types and subtypes with no representation in the classified app reviews as well as types with no mapping to guidelines. Consequently, only the Alternative Interaction, Contextual Notifications, Focus and Structure, Text Equivalents and User Input types were included in the questionnaire. Within these types, only the Automated Functionality subtype (from the Alternative Interaction type) was excluded.

Each question consisted in a bug description to which the respondents were asked to select, from the multiple choices available, the most suitable type or subtype to classify it. Based on existing bugs from both the iOS Bug Tracker and iOS App Directory, every question embraced one real bug description. However, after pretesting the questionnaire (described below), the urges to clarify some text and disconnect some descriptions from the original app in which they appeared were evident. The classification of the original bugs description was used as the correct answer, and the questionnaire hypothesis considers that the classification is more intuitive as the probability of responses identical to that classification increases.

The questionnaire was subdivided in 6 groups of questions. In the first section, the types were summarized and the respondents were asked to attribute a type to a given bug. In the following 4 sections, information about the subtypes of each of the aforementioned types (except User input, since it only has one subtype) was presented and respondents were asked to select the most appropriate subtype to the bug description. Every type and sub type was represented in the questions, in some cases more than once, and all 28 questions were mandatory.

Since we firstly found a proper type to describe the bug and only then scrutinized the details and found the sub type, this process mirrors the reasoning followed during the construction of the classification and the validation phase. Moreover, this prevented the survey from being too tiresome or time-consuming, which is supposed to inhibit the dropout rate. A section concerning the sociodemographic data of the respondents closed the questionnaire, with no mandatory questions. The objective of including this section was to enable us to detect some trends or patterns in case the answers diverged considerably from the desired intuitiveness.

A first version of the questionnaire was pretested with a subgroup of 2 users with no prior knowledge of the theme. Feedback on the difficulty level and the time needed was collected. The questionnaire was adjusted to be shorter, linguistically more accessible and retested favorably with another subset of 3 users. The questionnaire, as presented in Appendix D, was hosted in Google Forms and made available from May 17, 2021 to June 23, 2021. Once our target were screen reader users, the questionnaire was divulged online in blindness related groups and associations during such period - including AppleVis, Guide-Dog Owners and a blind programmers mailing list, among others.

5.6 Questionnaire Results

In total, 37 responses were registered during the questionnaires availability period. The list of questions grouped by section, each representing a bug, and their correct classification can be found in Appendix E.

The aforementioned questionnaire hypothesizes that the classification is more intuitive as the probability of responses identical to the classification of the original bug increases. To express a success measure, we defined a minimum lower limit of 50% and a higher limit of at least 85% for confidence intervals.

To calculate the probability of a correct answer, R's Exact Binomial Test function (binom.test) was used, computing, per question, the number of success cases and the number of trials (37).

In Appendix F, all questions grouped by section, the number of correct answers, the estimated probability of success and the upper and lower limits of the confidence interval, with a 95% confidence level, are presented.

From the 28 bugs respondents were asked to classify, 26 registered a confidence interval with a lower value greater than 50%.

The two bugs that exhibited a value inferior to 50% both reached a lower limit of 39.49% and 16 wrong answers. They were the following:

Q2: The Misspelled Words option in the Rotor does not work in the compose message text field;

Q3: To make adjustments on the equalizer knobs, you have to double tap and hold the desired knob and simulate a rotation motion as if you were turning a real knob. It can be done but you have to go by trial and error and read the new value after you perform this manual gesture.

Regarding Q2, for which the correct classification was Alternative interaction, the majority of wrong answers were User Input (12). The Misspelled Words option is a Rotor item that only appears when editing text. Therefore, respondents may consider more intuitive classifying all text editing problems as User Input.

No other pattern is discernible in the responses to Q2, as they do not seem to correlate with age, screen reader experience or nationality.

Meanwhile, concerning Q3, in which the correct answer was Alternative Interaction as well, the majority of wrong answers was also User Input (14). Since the question is related to a common scenario in music production, in which the described knobs mimic real knobs on physical amplifiers' equalizers, to a certain extent, it implies music knowledge that many users may not have. We surmise that respondents interpreted the change of knob values as the user inputing a new value, therefore classifying it as User Input.

By scrutinizing the distribution of Q3 answers per nationality, a pattern becomes clear. Within the 14 respondents that classified the problem as User Input, 12 were Latin languages native speakers. As a result, we assume the root of misinterpreting this more complex English written scenario may be related to cultural perceptions or the phrasing employed itself.

Besides the previously presented bugs, both with upper values of the confidence interval of 72%, the following bug recorded an upper limit of the confidence interval of 84% and 11 wrong answers:

Q6: When scanning a new document, VoiceOver won't give any feedback to help you align the page.

In this case, no pattern could be perceivable concerning the distribution of wrong responses and no correlation with sociodemographic aspects could be established.

On the contrary, the upper limit of the confidence interval is greater than 85% in 25 of the 28 bugs - surpassing our success criteria.

Furthermore, with the exception of Q2 (56%), Q3 (56%) and Q6 (70%), 25 out of the 28 bugs in analysis exhibit an estimated probability of success, or point estimate, greater than 75%. Moreover, also considering the fact that 23 questions present upper limits of the confidence interval above 90%, the level of concordance with the proposed classification is reinforced.

Consequently, we can conclude, with a confidence level of 95%, that the majority of screen reader users would classify these bugs correctly.

5.7 Discussion

The validation of the proposed classification was performed in two stages.

Firstly, the classification's completeness was assessed. A sample of the 640 theoretically least accessible apps from the AppleVis iOS App Directory was selected. The user submitted Accessibility Comments and Other Comments from each selected app were explored, and every described bug was classified according to the classification proposed in Chapter 4. From the 640 apps, 473 included detailed information that led to the identification of 793 bug instances.

Excluding the types Low Vision and Speech Output, which were practically inexistent in this sample due to their specific operative system nature, all other bug Types were recorded in this validation stage. Text Equivalents and Focus and Structure were the more predominantly identified bug types.

Notwithstanding, the emergence of completely inaccessible apps led to the creation of the a new bug type to express this complete lack of accessibility, named Unusable, and incorporating the subtype Completely Inaccessible. Since Unusable was the third most prominent bug type, this investigation attests that some developers still do not take accessibility into account.

The subtypes Unlabeled Elements, Unclear Labels and Unreachable Elements were the most abundant in the scrutinized app reviews, representing almost half of the classified bugs. These subtypes represent the bugs more easily relatable with the explored guidelines [16, 42] and the ones all testing tools detect [37]. Thus, this substantiates that developers are confused by the ambiguous nature of guidelines and the complexity of available tools [38]. Nonetheless, these bug subtypes are among the fastest to fix, according to the information from the iOS Bug Tracker extension (analyzed in Section 4.4).

The subtype Inaccessible Maps was also added to the Focus and Structure type, representing accessibility problems in maps that were not previously found in the iOS Bug Tracker data.

Consequently, we can answer the first research question positively:

RQ1: Can the proposed classification effectively cover vision accessibility errors reported by users in iOS apps?

Since the proposed classification was applied to the subset of the 640 least accessible apps on the AppleVis iOS App Directory, which comprises user submitted reviews, and was able to identify 793 bug instances, we conclude that it does indeed cover vision accessibility errors reported by users in iOS apps. Furthermore, by creating the Unusable type and also incorporating the Inaccessible Maps subtype on the Focus and Structure type, the proposed classification becomes even more effective in covering vision accessibility bugs on iOS apps.

The classification's intuitiveness was also evaluated.

A questionnaire was developed, presenting the classification to screen reader users and asking them to classify bug descriptions based on the available types and subtypes. On total, 28 bugs were described, with 5 types and 18 subtypes presented to respondents. The classification subsetting was based on the types' representativeness on guidelines and the subtypes' predominance on the previous validation stage.

Each question was based on the description of a bug previously classified from the AppleVis iOS Bug Tracker or App Directory. The original classification of the bug was considered the correct answer to each question.

The questionnaire received 37 responses. The number of correct answers was calculated per question and the confidence interval, with a confidence level of 95%, was computed for each one.

This investigation defined a confidence interval with a minimum lower limit of 50% and a higher limit of, at least, 85% as a measure of the questionnaire's intuitiveness.

Hence, we can address RQ2 approvingly:

RQ2: Is the proposed classification perceived similarly by several visually impaired users?

Once 26 out of the 28 questions presented a lower limit of the confidence interval superior to 50% - representing the majority of users - as well as the 25 questions exhibiting an upper limit of the confidence interval superior to 85% - indeed, 23 of them were greater than 90%, we conclude, with a 95% level of confidence, or a 5% level of significance, that the proposed classification is perceived similarly by several visually impaired users.

Furthermore, the aforementioned 25 questions displayed an estimated probability of success greater than 75%.

Notwithstanding, some of the questions failing to reach the established confidence interval provide some insights regarding improvements in the classification. In the two questions with

the lowest percentage of correct answers, respondents opted for the type User Input instead of Alternative Interaction.

In one of the questions (Q3), this may be due to linguistic reasons, as the bug description was particularly long and the majority of wrong answers was given by Latin languages native speakers. Besides, the question described a very specific music production scenario, and perhaps not all respondents were familiar with such terminology.

However, in the former question (Q2), the User Input answers can be attributed to the fact that the described functionality, the Misspelled Words Rotor item, is only available during text input. Thus, the classification's intuitiveness may be further improved by grouping into User Input all bugs related to text input.

Chapter 6

Conclusions

The present investigation arose from the need of a more user-centered approach to ensure that accessibility problems faced by disabled users were given the proper relevance [19].

Relating reported problems with the available accessibility guidelines requires expertise [19] and the existent tools to automatically verify the accessibility of applications only cover a low percentage of the available guidelines [31, 37]. Moreover, the existing guidelines do not take into account all problems reported by disabled users [3, 19], failing to promote the comprehension of such problems [19]. Thus, developers and testers do not find them very clear nor easy to understand [38].

Disabled users commonly report accessibility problems in written text - on feedback forms, email or in social media. However, the literature mentions a scarceness of accessibility focused reviews in app stores [5].

The mismatching between the terminology employed by users to report accessibility problems and the available guidelines further bewilders developers and testers [19], leading to social issues regarding accessibility [38]. Consequently, this leads to a minor focus on accessibility [38]and a lower priority in fixing these issues [19].

This investigation's focus on iOS vision accessibility led to the identification of the AppleVis repositories as excellent sources of information. Its publicly available iOS Bug Tracker and App Directory, to the best of our knowledge, were never explored in an academic context and their user focused bug descriptions and user submitted app reviews proved to be invaluable sources of information throughout this work.

By exploring and extending the AppleVis iOS Bug Tracker, as well as mapping the identified problems with the available accessibility guidelines, a classification of iOS vision accessibility bugs was proposed. The exploration of the available data, grouped by the proposed classification types and subtypes, provided a deeper comprehension of the severity and longevity of these kinds of bugs on iOS.

Once the proposed classification was based on user-centered information and classified all bugs in scrutiny, we conclude that it typifies iOS accessibility bugs reported by visually impaired users. Moreover, as the WCAG and BBC guidelines were used to group the bugs into Types, we also surmise that the classification conforms to the available accessibility guidelines.

Hence, our thesis' hypothesis is validated.

The completeness and intuitiveness of the proposed classification were validated. The 640 least accessible app reviews on the AppleVis iOS App Directory were scrutinized and 793 bug instances were categorized according to the proposed classification. Two additions were made to the classification. Namely, the subtypes Completely Inaccessible - classified with the type Unusable, and Inaccessible Maps - in the Focus and Structure type, were incorporated into the classification. Therefore, the proposed classification effectively covered vision accessibility errors reported by users in iOS apps, approvingly answering RQ1, and reinforcing its completeness.

The classification's intuitiveness was tested through a questionnaire, in which 37 respondents classified 28 bug descriptions. With a confidence level of 95%, 26 of the 28 bugs exhibited a lower limit of the confidence interval above 50% and 25 of those displayed an upper limit superior to 85%. Thus, we conclude that the majority of screen reader users would classify these bugs correctly, with a confidence level of 95%.

Accordingly, the proposed classification is perceived similarly by several visually impaired users, answering favorably to RQ2.

6.1 Practical Contributions

By validating an user-centered classification of iOS vision accessibility problems that conforms to the available accessibility guidelines and is both complete and intuitive, this work improves the current tools available to both users and developers.

On one hand, users gain the possibility of categorizing their bug reports according to a classification based on terms they frequently employ. On the other hand, developers who receive unclassified accessibility bug reports are also able to explore the classification and feasibly deduce in which type and subtype each bug can be framed.

A proper classification contributes to the developers' comprehension of bugs, facilitating their job of relating them with the existing guidelines, as well as to establish connections to past bugs previously faced or solved. Consequently, fixing accessibility bugs becomes faster and easier [19], benefiting developers and users simultaneously.

6.2 Theoretical Contributions

The present work ameliorates the state of the art in accessibility, by proposing and validating an user-centered classification able to relate with the available guidelines. To the best of our knowledge, no previous study had proposed a classification of accessibility bugs based on the terminology employed by a specific disability group to describe problems they frequently face.

Furthermore, this terminology was obtained in written format, replicating the most common way in which disabled users submit bug reports. This ensures that developers will encounter a terminology similar to one employed in the accessibility bug reports they receive. In turn, social issues regarding accessibility can be minimized [38], decreasing the time needed to fix bugs [19].

By resorting to data available on AppleVis for the consecution of this investigation, we aimed to fill the gap in the literature denoted by Alshayban et al. [5] and, ultimately, revealed the usefulness of the information available at disability related repositories.

6.3 Limitations

Nevertheless, some limitations can be identified in this investigation.

The focus on vision accessibility restricts the impact of the proposed classification to visually disabled users or researchers on the area. Indeed, its transposition to accessibility problems faced by other disability groups is not easily attainable.

Although some vision accessibility problems are similarly experienced on other operating systems, touch or mouse-based, there are specificities of touch-based operating systems - some of them regarding iOS in particular (e.g. Rotor), that may limit the scope of application of the proposed classification to iOS. Furthermore, as stated in Chapter 3.2, the data in AppleVis included predominantly VoiceOver bugs, with Low Vision problems being noticeably less represented. Thus, the proposed classification reflects this bias and emphasizes VoiceOver bugs while displaying less details and instances of Low Vision related problems.

Due to the shortage of time, the validation stage did not encompass testing the proposed classification with developers. Considering that the questionnaire was shared in a forum of blind developers, some of the respondents may have been developers, as well as screen reader users. Notwithstanding, the questionnaire was created assuming screen reader knowledge and it would not be suitable for developers that lacked such knowledge.

Once one of the objectives of the investigation was to test how intuitive the questionnaire was, we opted to enquire screen reader users first, which exhausted the time and possibilities to develop an usefulness test to assess how the classification could facilitate developers' jobs.

6.4 Future Work

This investigation can be refined in future works.

For instance, the classification may be improved with the findings of the questionnaire, specially in what concerns classifying all text editing related bugs as User Input. Although the majority of users classified the second question, regarding the Misspelled Words option in the Rotor, correctly, the high percentage of User Input responses suggests that the classification's intuitiveness could be further enhance with this change. Thus, performing this change and developing a new questionnaire to uphold its impact would be an immediate future development of this work.

To test the classification's intuitiveness in conditions similar to the ones users face on their usual interaction with technology, we believe developing an app with purposely injected accessibility bugs would represent an amelioration of the questionnaire method employed in this investigation. This app would present users with the materialization of the bugs described on the questionnaire, or other more complex ones, and allow users to report them using the proposed classification. This would eliminate possible language barriers and allow users to experience the bugs directly, and then, replicate the common process of reporting bugs or writing reviews on the AppleVis App Directory.

Another suggestion for future work concerns the creation of an open-source code repository with implementation examples of non-purposeful accessibility bugs and proposals for fixes. This repository would be structured based on the proposed classification and, for each bug subtype, contain one or more code examples portraying how the bug usually occurs. This extension of the classification would supply a valuable resource to developers, since they would be able, after identifying a bug's type and subtype, to explore possible fixes, improving and accelerating the bug fixing process, therefore enhancing accessibility for all users.

Finally, we also believe this investigation should be replicated on other platforms, such as Android, macOS, Chrome OS or Windows. Concomitantly, applying the same methodology on wider repositories of Low Vision problems, as well as to other disability groups - auditory; cognitive, learning and neurological; physical; speech; disabilities [15] - would also represent a great opportunity for making technology more accessible to everyone.

Appendix A

AppleVis Original Data

The linked Excel file comprises the original data from the AppleVis iOS Bug Tracker and App Directory.

https://docs.google.com/file/d/1bDkjzzVAsUZSwKpNNabwkN0XYazQQvIR/edit?usp=docslist_api&filetype=msexcel

Appendix B

Bug Tracker Extension and Classification

The linked Excel file contains the list of all Bug Tracker bugs and the respective new values created during the Bug Tracker extension. It also comprises other sheets summarizing all the new attributes' possible values, namely System Function; Problems; the mapping process to create the Type; and Subtype.

https://docs.google.com/file/d/1FiOx_Ze3iX-cMjQjIkIcaS4ADG7qqzPS/edit?usp=docslist_api&filetype=msexcel

Appendix C

App Directory Bugs and Respective Classifications

The linked Excel file presents the subset of App Directory app reviews and their classification. The "Original Classification" sheet contains the validation process as it was performed, in some cases containing multiple classifications per app, while the "Processed Bugs" sheet contains the result of the analysis script, presenting the list of all identified bug subtypes.

https://docs.google.com/file/d/1NGcOCEw9kM2p2cCArd5vTT7UH7KA735b/edit?usp=docslist_api&filetype=msexcel

Appendix D

Questionnaire

Validation of a Vision Accessibility Bugs Classification

This questionnaire is part of an academic investigation.

My goal with this work is to bridge the gap between the language employed by users to report vision accessibility bugs and the guidelines available to developers. To do so, I am proposing a new categorization of iOS vision accessibility bugs.

Now, I need your help to test if everyone can classify the bug descriptions in the same way.

In the following sections, firstly, I will present you the possible options and an explanation for each. Then, given a list of bug descriptions, you are asked to select the option you consider more adequate to classify each one.

Filling this questionnaire takes approximately 10 minutes and the only requirement for answering is to have ever used a screen reader.

In case you would like to receive the results of this study or if you need help to clarify any question, please contact me at up200802385@edu.fe.up.pt.

Thanks for your collaboration!

Diogo Melo
Masters in Software Engineering - Faculty of Engineering of University of Porto (FEUP)

* Required

1. All data is confidential and used for the purpose of this study only. You are free to cease your participation at any moment. *

Mark only one oval.

- ◯ I agree to voluntarily participate in the study

Categories

The 5 main categories are as follows:

1 - Alternative interaction
Problems with either:
- Keyboard navigation;
- Braille navigation and input;
- Alternatives to complex gestures (e.g. drag and drop) or;
- Fast navigation options (e.g. navigate by Lines, Misspelled words, links).

2 - Contextual notifications
When activated actions or displayed visual cues are not properly reported by spoken announcements or sound hints.

3 - Focus and structure
Either when:
- Available elements can not be focused;
- Unavailable elements can be focused;
- Items can not be activated or;
- It is not possible to navigate the interface as expected (e.g. change page or navigate by heading)

4 - Text equivalents
Unclear, inexistent or inaccurate labels of non textual elements (e.g. buttons, images, badges).

5 - User input
Problems related to the input of text or other information formats (e.g. pickers, sliders).

2. Despite being available, the Save button can not be found with VoiceOver. *

Mark only one oval.

- ◯ Alternative interaction
- ◯ Contextual notifications
- ◯ Focus and structure
- ◯ Text equivalents
- ◯ User input

3. The Misspelled words option in the Rotor does not work in the compose message text field. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

4. To make adjustments on the equalizer knobs, you have to double tap and hold the desired knob and simulate a rotation motion as if you were turning a real knob. It can be done but you have to go by trial and error and read the new value after you perform this manual gesture. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

5. VoiceOver will read several Headings, but if you try to navigate using the Headings option in the Rotor VoiceOver won't scroll past the page visible on the screen. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

6. When inserting the name of a new contact, the usual keyboard typing options do not work and you have to double tap each letter to insert it. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

7. When scanning a new document, VoiceOver won't give any feedback to help you align the page. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

8. You can define your status as Online, Away or Do not disturb. However, there is no label for the selected status. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

9. You can't read an email using a Braille display. *

Mark only one oval.

- Alternative interaction
- Contextual notifications
- Focus and structure
- Text equivalents
- User input

Focus & Structure

In this category you have 5 options, representing some frequent scenarios.

1 - Extraneous elements
If VoiceOver allows the selection of elements that are not visible.

2 - Focus
When the VoiceOver cursor moves to an element without the user's intent.

3 - Item activation
When you perform the activation gesture but the item is not activated.

4 - Navigation
When navigation controls don't work as expected (e.g. heading navigation, page navigation).

5 - Unreachable elements
When Voiceover can not detect an element present in the user interface.

10. You can not activate the Save button with a double tap. *

Mark only one oval.

- Extraneous elements
- Focus
- Item activation
- Navigation
- Unreachable elements

11. You can not find the controls to bookmark a page with VoiceOver turned on. *

Mark only one oval.

- Extraneous elements
- Focus
- Item activation
- Navigation
- Unreachable elements

12. When selecting a payment method, the VoiceOver cursor moves to the first listed method every time you flick. *

Mark only one oval.

- Extraneous elements
- Focus
- Item activation
- Navigation
- Unreachable elements

13. The playlist selection page is properly labeled. However, navigating by heading is unreliable. *

Mark only one oval.

- Extraneous elements
- Focus
- Item activation
- Navigation
- Unreachable elements

14. VoiceOver finds and reads elements on the Profile tab which are not visually present or actionable. *

Mark only one oval.

- ◯ Extraneous elements
- ◯ Focus
- ◯ Item activation
- ◯ Navigation
- ◯ Unreachable elements

15. Sometimes, when you swipe, the VoiceOver cursor won't advance and, other times, it ignores the following item, skipping to the next one. *

Mark only one oval.

- ◯ Extraneous elements
- ◯ Focus
- ◯ Item activation
- ◯ Navigation
- ◯ Unreachable elements

Contextual Notifications

There are only two choices in this category:

1 - Announcements
Absence of spoken announcements when an action is performed or some written cue is updated.

2 - Sound hints
Absence of sound hints that replace visual indicators (e.g. when the device is unlocked with biometric authentication).

16. During a game, VoiceOver doesn't announce when your opponent plays and you have to scan the whole board to find out what piece was moved and to where. *

Mark only one oval.

- ◯ Announcements
- ◯ Sound hints

17. VoiceOver won't play a sound when you start to reorder your reminders. *

Mark only one oval.

- Announcements
- Sound hints

18. When taking a photo to create a new avatar, VoiceOver won't automatically read the cues to position yourself. You have to constantly touch the bottom of the screen to get updated feedback. *

Mark only one oval.

- Announcements
- Sound hints

Alternative Interaction

The five options in this category are the following:

1 - Braille
Any problem related to reading or navigating while using a Braille display.

2 - Braille typing
Related to text input with Braille displays or Braille Screen Input.

3 - Fast navigation
Rotor options are having an inconsistent behaviour when navigating between dynamic interface elements (e.g. navigate by Lines, Misspelled words, links).

4 - Gesture alternatives
Inexistent or flawed Rotor actions to replace complex gestures (e.g. drag and drop).

5 - Keyboard navigation
Problems when navigating via external keyboard.

19. During a game, the only way to move your pieces in the board is to double tap and hold and then drag them to where you want to play. *

Mark only one oval.

- Braille
- Braille typing
- Fast navigation
- Gesture alternatives
- Keyboard navigation

20. When you are navigating with a Braille display, the AI generated GIF description is not presented, despite being read by VoiceOver. *

Mark only one oval.

- Braille
- Braille typing
- Fast navigation
- Gesture alternatives
- Keyboard navigation

21. When reading a post, you can not navigate by Lines in the Rotor options. *

Mark only one oval.

- Braille
- Braille typing
- Fast navigation
- Gesture alternatives
- Keyboard navigation

22. You can not type a reply to a comment with a Braille display. *

Mark only one oval.

- Braille
- Braille typing
- Fast navigation
- Gesture alternatives
- Keyboard navigation

23. You can not activate the Send button with an external keyboard. *

Mark only one oval.

- Braille
- Braille typing
- Fast navigation
- Gesture alternatives
- Keyboard navigation

Text Equivalents

In this final category, there are six options regarding specific scenarios of improper or incomplete element labeling.

1 - Incomplete information
VoiceOver reads some but not all the relevant visually available information.

2 - Label with extraneous text
The element is clearly labeled but some non-informative extra text is part of the label.

3 - Unclear label
When the label is not clear about what the item represents or what action it triggers.

4 - Unlabeled element
When an element is not labeled and users can not know its intent.

5 - Unspoken selection
When Voiceover reports the element but not its selection status (e.g. check boxes, multi selection menus).

6 - Wrong label
The label is clear but represents erroneously the type, status or action of the item.

24. The buttons for the modulation effects don't have labels. *

Mark only one oval.

- Incomplete information
- Label with extraneous text
- Unclear label
- Unlabeled element
- Unspoken selection
- Wrong label

25. The Stop button is read as Pause. *

Mark only one oval.

- Incomplete information
- Label with extraneous text
- Unclear label
- Unlabeled element
- Unspoken selection
- Wrong label

26. VoiceOver will read "img_meteorstatus_" before each weather forecast for the next 24 hours. *

Mark only one oval.

- Incomplete information
- Label with extraneous text
- Unclear label
- Unlabeled element
- Unspoken selection
- Wrong label

27. When choosing your opponent, VoiceOver will read the username and country but won't read the player name, age and ranking that is visually present on the screen. *

Mark only one oval.

- ◯ Incomplete information
- ◯ Label with extraneous text
- ◯ Unclear label
- ◯ Unlabeled element
- ◯ Unspoken selection
- ◯ Wrong label

28. Before making a call, there are options to select audio or video but there is no feedback about which one is selected. *

Mark only one oval.

- ◯ Incomplete information
- ◯ Label with extraneous text
- ◯ Unclear label
- ◯ Unlabeled element
- ◯ Unspoken selection
- ◯ Wrong label

29. When you want to share your photos, you have to change to Rotor Character navigation to understand the sharing options, because the buttons are labeled as "share" followed by the first letter of the social network, like "sharet", "sharey", "sharei" and "sharef". *

Mark only one oval.

- ◯ Incomplete information
- ◯ Label with extraneous text
- ◯ Unclear label
- ◯ Unlabeled element
- ◯ Unspoken selection
- ◯ Wrong label

Sociodemographic Data

Thanks for your help!

The following questions are not mandatory but they would help me to understand some trends in the results. The data will not be treated individually. The processing will be purely statistical.

If you do not wish to answer them, please proceed to the next page and submit.

30. Where are you from?

_____

31. What is your age?

_____

32. What mobile screen reader do you use?

Mark only one oval.

- ◯ I don’t use any mobile screen reader
- ◯ VoiceOver on iOS
- ◯ TalkBack on Android
- ◯ Other: _____

33. For how long have you been using a screen reader?

Mark only one oval.

- Less than 1 year
- 1 - 3 years
- 4 - 6 years
- 7 - 10 years
- More than 10 years

Thanks for your collaboration :)

Your participation is very important!

If you know anyone who may also contribute, please don't hesitate to share with them.

Please let me know if you have any doubts or suggestions to improve this classification (e.g. more categories, name changes, …). Leave your comments below or reach me at up200802385@edu.fe.up.pt.

34.

Appendix E

Questionaire Correct Answers

Table E.1: Questionaire Correct Answers

#	Section	Question	Correct Answer
Q1	Types	Despite being available, the Save button can not be found with VoiceOver.	Focus & Structure
Q2		The Misspelled words option in the Rotor does not work in the compose message text field.	Alternative Interaction
Q3		To make adjustments on the equalizer knobs, you have to double tap and hold the desired knob and simulate a rotation motion as if you were turning a real knob. It can be done but you have to go by trial and error and read the new value after you perform this manual gesture.	Alternative Interaction
Q4		VoiceOver will read several Headings, but if you try to navigate using the Headings option in the Rotor VoiceOver won't scroll past the page visible on the screen.	Focus & Structure
Q5		When inserting the name of a new contact, the usual keyboard typing options do not work and you have to double tap each letter to insert it.	User Input
Q6		When scanning a new document, VoiceOver won't give any feedback to help you align the page.	Contextual Notifications
Q7		You can define your status as Online, Away or Do not disturb. However, there is no label for the selected status.	Text Equivalents
Q8		You can't read an email using a Braille display.	Alternative Interaction
Q9	Focus & Structure	You can not activate the Save button with a double tap.	Item Activation
Q10		You can not find the controls to bookmark a page with VoiceOver turned on.	Unreachable Elements
Q11		The playlist selection page is properly labeled. However, navigating by heading is unreliable.	Navigation
Q12		VoiceOver finds and reads elements on the Profile tab which are not visually present or actionable.	Extraneous Elements
Q13		When selecting a payment method, the VoiceOver cursor moves to the first listed method every time you flick.	Focus
Q14		Sometimes, when you swipe, the VoiceOver cursor won't advance and, other times, it ignores the following item, skipping to the next one.	Focus
Q15	Contextual Notifications	During a game, VoiceOver doesn't announce when your opponent plays and you have to scan the whole board to find out what piece was moved and to where.	Announcements
Q16		VoiceOver won't play a sound when you start to reorder your reminders.	Sound Hints
Q17		When taking a photo to create a new avatar, VoiceOver won't automatically read the cues to position yourself. You have to constantly touch the bottom of the screen to get updated feedback.	Announcements
Q18	Alternative Interaction	During a game, the only way to move your pieces in the board is to double tap and hold and then drag them to where you want to play.	Gesture Alternatives
Q19		When you are navigating with a Braille display, the AI generated GIF description is not presented, despite being read by VoiceOver.	Braille
Q20		When reading a post, you can not navigate by Lines in the Rotor options.	Fast Navigation
Q21		You can not type a reply to a comment with a Braille display.	Braille Typing
Q22		You can not activate the Send button with an external keyboard.	Keyboard Navigation
Q23	Text Equivalents	The buttons for the modulation effects don't have labels.	Unlabeled Element
Q24		The Stop button is read as Pause.	Wrong Label
Q25		VoiceOver will read "img_meteorstatus_" before each weather forecast for the next 24 hours.	Label with Extraneous Text
Q26		When choosing your opponent, VoiceOver will read the username and country but won't read the player name, age and ranking that is visually present on the screen.	Incomplete Information
Q27		Before making a call, there are options to select audio or video but there is no feedback about which one is selected.	Unspoken Selection
Q28		When you want to share your photos, you have to change to Rotor Character navigation to understand the sharing options, because the buttons are labeled as "share" followed by the first letter of the social network, like "sharet", "sharey", "sharei" and "sharef".	Unclear Label

Appendix F

Questionnaire Exact Binomial Test Results

Table F.1: Questionnaire Exact Binomial Test Results

#	Section	Correct	Confidence Interval (Upper)	Point Estimate	Confidence Interval (Lower)
Q1	Types	28	88.23%	75.68%	58.80%
Q2		21	72.90%	56.76%	39.49%
Q3		21	72.90%	56.76%	39.49%
Q4		33	96.97%	89.19%	74.58%
Q5		31	93.81%	83.78%	67.99%
Q6		26	84.13%	70.27%	53.02%
Q7		29	90.17%	78.38%	61.79%
Q8		31	93.81%	83.78%	67.99%
Q9	Focus & Structure	32	95.46%	86.49%	71.23%
Q10		28	88.23%	75.68%	58.80%
Q11		36	99.93%	97.30%	85.84%
Q12		32	95.46%	86.49%	71.23%
Q13		33	96.97%	89.19%	74.58%
Q14		31	93.81%	83.78%	67.99%
Q15	Contextual Notifications	34	98.30%	91.89%	78.09%
Q16		33	96.97%	89.19%	74.58%
Q17		29	90.17%	78.38%	61.79%
Q18	Alternative Interaction	33	96.97%	89.19%	74.58%
Q19		35	99.34%	94.59%	81.81%
Q20		32	95.46%	86.49%	71.23%
Q21		30	92.04%	81.08%	64.84%
Q22		34	98.30%	91.89%	78.09%
Q23	Text Equivalents	33	96.97%	89.19%	74.58%
Q24		33	96.97%	89.19%	74.58%
Q25		32	95.46%	86.49%	71.23%
Q26		33	96.97%	89.19%	74.58%
Q27		30	92.04%	81.08%	64.84%
Q28		31	93.81%	83.78%	67.99%

References

[1] P. Acosta-Vargas, L. Salvador-Ullauri, J. Jadán-Guerrero, C. Guevara, S. Sanchez-Gordon, T. Calle-Jimenez, P. Lara-Alvarez, A. Medina, and I. L. Nunes. Accessibility Assessment in Mobile Applications for Android. In Isabel Nunes, editor, *Advances in Human Factors and Systems Interaction. AHFE 2019. Advances in Intelligent Systems and Computing*, volume 959. Springer, Cham, 2020. doi:10.1007/978-3-030-20040-4_25.

[2] A. A. Al-Subaihin, A. S. Al-Khalifa, and H. S. Al-Khalifa. Accessibility of Mobile Web Apps by Screen Readers of Touch-Based Mobile Phones. In M. Matera and G. Rossi, editors, *Trends in Mobile Web Information Systems. MobiWIS 2013. Communications in Computer and Information Science*, volume 183. Springer, Cham, 2013. doi:10.1007/978-3-319-03737-0_5.

[3] 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. *Univ Access Inf Soc*, 2021. doi:10.1007/s10209-020-00785-w.

[4] G. Alexiou. *Largest U.S. Blind Advocacy Group Bans Web Accessibility Overlay Giant AccessiBe*. Forbes. Retrieved June 27, 2021, from https://cutt.ly/SmiuJPP, June 2021.

[5] A. Alshayban, I. Ahmed, and S. Malek. Accessibility Issues in Android Apps: State of Affairs, Sentiments, and Ways Forward. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, pages 1323–1334, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3377811.3380392.

[6] W. T. Andrade, R. G. de Branco, M. I. Cagnin, and D. M. B. Paiva. Incorporating Accessibility Elements to the Software Engineering Process. *Advances in Human-Computer Interaction*, 2018. doi:10.1155/2018/1389208.

[7] Apple. *Vision. For Every Point of View*. Retrieved January 8, 2021, from https://www.apple.com/accessibility/vision/.

[8] Apple Developer. *Human Interface Guidelines. Accessibility*. Retrieved January 8, 2021, from https://developer.apple.com/design/human-interface-guidelines/accessibility/overview/introduction/.

[9] Apple Developer. *Swift*. Retrieved January 8, 2021, from https://developer.apple.com/swift/.

[10] Apple Developer. *Type Property isSelected*. Retrieved January 8, 2021, from https://developer.apple.com/documentation/swiftui/accessibilitytraits/isselected.

[11] Apple Developer. *User Interface Testing*. Retrieved January 8, 2021, from https://developer.apple.com/library/archive/documentation/DeveloperTools/Conceptual/testing_with_xcode/chapters/09-ui_testing.html.

[12] AppleVis. *iOs and iPadOS Apps*. Retrieved January 8, 2021, from https://www.applevis.com/apps/ios/browse.

[13] AppleVis. *The AppleVis Bug Tracker*. Retrieved January 8, 2021, from https://www.applevis.com/bugs.

[14] AppleVis. *Welcome to AppleVis*. Retrieved January 8, 2021, from https://www.applevis.com/.

[15] A. Bai, K. Fuglerud, R. Skjerve, and T. Halbach. Categorization and Comparison of Accessibility Testing Methods for Software Development. *Studies in Health Technology and Informatics*, 256:821–831, 2018.

[16] BBC. *Accessibility for Products*. Retrieved January 4, 2021, from https://www.bbc.co.uk/accessibility/forproducts/guides/mobile/summary/.

[17] Blank Rome. *New Ruling Reiterates That Websites and Mobile Apps Need to Be ADA Compliant*. Retrieved January 11, 2021, from https://cutt.ly/amiuBKB, January 2019.

[18] M. C. N. Carvalho, F. S. Dias, A. G. S. Reis, and A. P. Freire. Accessibility and Usability Problems Encountered on Websites and Applications in Mobile Devices by Blind and Normal-Vision Users. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, SAC '18, pages 2022–2029, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3167132.3167349.

[19] R. Clegg-Vinell, C. Bailey, and V. Gkatzidou. Investigating the Appropriateness and Relevance of Mobile Web Accessibility Guidelines. In *Proceedings of the 11th Web for All Conference*, W4A '14, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2596695.2596717.

[20] G. Dean. *Apple Could Cut, or Raise, Executive Bonuses by 10 per cent Based on their Performance on Environmental and Social Issues in 2021*. Business Insider. Retrieved January 11, 2021, from https://cutt.ly/8miu42I, January 2021.

[21] Deque. *EU Web Accessibility Compliance and Legislation*. Retrieved January 11, 2021, from https://www.deque.com/blog/eu-web-accessibility-compliance-and-legislation/, January 2020.

[22] D.E. Dilger. *Apple VoiceOver Accessibility receives award from American Foundation for the Blind*. AppleInsider. Retrieved January 10, 2021, from https://cutt.ly/Qmiiqf4, May 2015.

[23] 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)*, pages 116–126, 2018. doi:10.1109/ICST.2018.00021.

[24] European Commission. *United Nations Convention on the Rights of Persons with Disabilities*. Retrieved January 11, 2021, from https://ec.europa.eu/social/main.jsp?catId=1138&langId=en.

[25] GitHub. *KIF*. Retrieved January 10, 2021, from https://github.com/kif-framework/KIF.

[26] E. G. Hansen, R. J. Mislevy, L. S. Steinberg, M. J. Lee, and D. C. Forer. Accessibility of Tests for Individuals with Disabilities within a Validity Framework. *System*, 33(1):107–133, 2005. doi:10.1016/j.system.2004.11.002.

[27] V.-V. Helppi. *Getting Started with KIF for Functional iOs UI Testing*. BitBar. Retrieved January 10, 2021, from https://bitbar.com/blog/getting-started-with-kif-for-functional-ios-ui-testing/.

[28] J. A. Krosnick. Survey Research. *Annual Review of Psychology*, 50(1):537–567, 1999. doi:10.1146/annurev.psych.50.1.537.

[29] J. Lewkowicz. *WWDC: Apple introduces new way to build UIs using Swift*. SD Times. Retrieved January 8, 2021, from https://sdtimes.com/apple/wwdc-apple-introduces-new-way-to-build-uis-on-apple-using-swift/, June 2019.

[30] S. Luján-Mora and F. Masri. Integration of Web Accessibility into Agile Methods. In *ICEIS 2012 - Proceedings of the 14th International Conference on Enterprise Information Systems*, 2012. doi:10.5220/0004095001230127.

[31] D. A. Mateus, C. A. Silva, M. M. Eler, and A. P. Freire. Accessibility of Mobile Applications: Evaluation by Users with Visual Impairment and by Automated Tools. In *Proceedings of the 19th Brazilian Symposium on Human Factors in Computing Systems*, IHC '20, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3424953.3426633.

[32] J. M. Mucha. Combination of Automatic and Manual Testing for Web Accessibility. Master's thesis, University of Agder, 2018. URL: https://uia.brage.unit.no/uia-xmlui/handle/11250/2563326.

[33] U. Paz. *Weaving Web Accessibility With Usability*. Smashing. Retrieved January 13, 2021, from https://www.smashingmagazine.com/2020/11/weaving-web-accessibility-usability/, November 2020.

[34] S. Sanchez-Gordon and S. Luján-Mora. A Method for Accessibility Testing of Web Applications in Agile Environments. *7th World Congress for Software Quality (WCSQ 2017). Lima, Peru.*, mar, 2017. URL: https://www.researchgate.net/publication/318214191_A_Method_for_Accessibility_Testing_of_Web_Applications_in_Agile_Environments.

[35] L. C. Serra, L. P. Carvalho, L. P. Ferreira, J. B. S. Vaz, and A. P. Freire. Accessibility Evaluation of E-Government Mobile Applications in Brazil. *Procedia Computer Science*, 67:348–357, 2015. doi:10.1016/j.procs.2015.09.279.

[36] M. Shcheglov. *UI Testing iOS Application with EarlGrey*. On Swift Wings. Retrieved January 8, 2021, from https://www.onswiftwings.com/posts/ui-tests-earlgrey/#:~:text=EarlGrey%20is%20a%20white%2Dbox,maintain%20(no%20waiting%20clauses), May 2020.

[37] 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*, DSAI 2018, pages 286–293, New York, NY, USA, 2018. Association for Computing Machinery. doi:10.1145/3218585.3218673.

[38] L. Steen-Hansen and S. Fagernes. The Importance of Process-Oriented Accessibility Guidelines for Web Developers. *Studies in Health Technology and Informatics*, 229:439–449, 2016.

[39] M.-L. Sánchez-Gordón and L. Moreno. Toward an Integration of Web Accessibility into Testing Processes. *Procedia Computer Science*, 27:281–291, 2014. doi:10.1016/j.procs.2014.02.031.

[40] W3C. *Web Content Accessibility Guidelines WCAG 2.0*. Retrieved January 2, 2021, from https://www.w3.org/TR/WCAG20/, December 2008.

[41] W3C. *Mobile Accessibility: How WCAG 2.0 and Other W3C/WAI Guidelines Apply to Mobile*. Retrieved January 2, 2021, from https://www.w3.org/TR/mobile-accessibility-mapping/, February 2015.

[42] W3C. *Web Content Accessibility Guidelines WCAG 2.1*. Retrieved January 9, 2021, from https://www.w3.org/TR/WCAG21/, June 2018.

[43] W3C. *W3C Accessibility Standards Overview*. Retrieved January 2, 2021, from https://www.w3.org/WAI/standards-guidelines/, April 2021.

[44] R. Whitaker. *A11yUITests: An XCUI Testing Library for Accessibility*. MobileA11y. Retrieved January 10, 2021, from https://mobilea11y.com/blog/a11yuitests/.

[45] World Health Organization. *Global Data on Visual Impairment 2010*. Retrieved January 10, 2021, from https://www.who.int/news-room/fact-sheets/detail/blindness-and-visual-impairment.