Accessibility Features of Developmental Platforms: Towards Developing Accessible Mobile Applications with Cross-Platform, Research Challenges and Opportunities

Muhammad Shoaib[1*] Donal Fitzpatrick[2] and Ian Pitt[1]
[1]School of Computer Science and Information Technology, University College Cork, Ireland.
[2]Centre for Excellence in Universal Design, National Disability Authority, Ireland
[*]muhammad.shoaib@cs.ucc.ie

Abstract

Mobile application development is a process of designing and developing smartphone-based applications for several platforms i.e., Android, iOS, etc. Mobile application development platforms can be categorised as native and cross-platform, each offering resources, frameworks, and tools to developers so they can develop smartphone-based applications using them. Native platforms concentrate on specific applications designed expressly for a particular platform i.e., Android or iOS. Cross-platform allows developers to develop applications with a single codebase that can work across multiple platforms. Examples include Xamarin and Unity. In this study, we have compared the accessibility features offered by Xamarin and Unity with those offered by native platforms. We analyzed the accessibility features offered by the iOS and Android platforms to determined whether and to what extent Xamarin and Unity offer the same accessibility capabilities. This analysis shows that numerous functionalities are shared by native iOS and Android APIs, however, some of them are not included in Xamarin and Unity. They also do not provide the implementation of fundamental APIs. This will require additional work by developers to write platform-specific code in native APIs to access the unavailable APIs. We have provided a comparative analysis of these platforms. Important accessibility aspects are highlighted that may prove useful for researchers and developers who are working to create accessible apps.

1 Introduction and Background

Android's popularity has climbed sharply since Google released version 1.0 of the operating system on September 23, 2008. With the release of Android 13 in August 2022, Google claimed to have around 3.6 billion active users worldwide [1]. According to Statista's 2023 report, the Google Play Store, the official digital retailer for Android, holds a 71.47% global market share for mobile operating systems. iOS is the other operating system that competes with Android for users. Since its June 2007 release, iOS has increased its market share to 27.6% [2]. This is why developers are keen to create apps that can work on both iOS and Android [3]. Choosing the platforms on which a mobile app will operate is the first step in the development process. Next, using platform-specific toolkits like the Android SDK, developers must create tailored solutions (i.e., native apps) for each selected platform. An app that functions across two or more mobile platforms is known as a cross-platform mobile application. Xamarin and Unity are important cross-platform toolkits that make mobile applications easier and lower development and maintenance expenses[3].

[*] Masterminded EasyChair and created the first stable version of this document

Xamarin was first introduced in 2011 as a cross-platform app development framework, Microsoft later purchased it, giving it more legitimacy than before. It provides access to native APIs as well as the potential for code reuse and sharing with other platforms. It is one of the most widely used and favored mobile development platforms. This cross-platform uses the C# programming language to produce mobile applications that perform remarkably well across all mobile devices [4]. Unity is a cross-platform game engine that was introduced in June 2005. It provides an interface that offers access to all development tools in a single location and is called an Integrated Development Environment (IDE). Developers can drag and drop objects into interfaces with the help of a visual editor and can change their properties. Unity uses an editor of your choice when it comes to coding. Microsoft's Visual Studio, which integrates most of the time, is the most common choice. Unity uses a language known as C#. All of the scripting languages that Unity uses are object-oriented [5].

Mobile applications are increasingly becoming an integral part of our daily life and the accessibility factor of native and cross-platform apps is essential for disabled people. It means making these apps useful for as many individuals as possible. Applying fairness to all people by granting them the same opportunities regardless of their abilities by adding additional features like scalable texts and screen reader support. Previous studies showed that developers should design the new mobile application by considering the accessibility factor [6-9]. Based on WHO more than 1.3 billion people worldwide have a disability which means 16% of the world's population, or one in six people[10].

This article presents a detailed analysis of mobile application development platforms and discusses their strengths and weaknesses with regard to accessibility. We have demonstrated the functionality of these platforms by exploring their accessibility features, plugins and extensions. We have listed accessibility APIs, tools, and packages and compared their features. We examined how cross-platform mobile applications wrap native APIs. We demonstrate that two of the most popular mobile cross-platform toolkits, Xamarin and Unity, cover around half and one-third of the screen reader-related iOS and Android APIs, respectively. Finally, we present our findings and talk about how we were able to use them to develop accessible mobile applications.

2 Most Used Tools for Mobile Applications Development

The quote "the right tools for the right job" has been around forever, and it holds for software development just as much as it does for any other field. These days there are several mobile app development platforms available. Selecting the appropriate mobile development platform is essential for designing a proper solution to make it scalable, accessible and maintainable. The cost and time of creating a mobile app may be reduced using the right development platform [11]. This section analyzes all four developmental platforms (Android, iOS, Xamarin and Unity) and briefly describes them.

2.1 Comparative Analysis of Native and Cross-platform Toolkits

Mobile application development through native platforms is very common. Every native platform has its programming language, like Java and Kotlin, the official programming language for the Android platform. On the other hand, having a good grasp of languages like Swift and Objective-C is essential for creating iOS applications. Cross-platform mobile application development can run on multiple platforms and we can also achieve the performance of Native platforms by using them [12]. Here, we compared Native (Android and iOS) and Cross-platforms (Xamarin and Unity). Table 1 provides detailed information about the strengths and Weaknesses of these platforms.

Platforms Name	Strengths	Weaknesses
Android Studio	Provide a user-friendly development environment Have excellent support for Android devices Supports various programming languages, including Java, Kotlin, and C++ Provide support to drag-and-drop tools. Previewing the app on different device sizes Better debugging and testing support Support profiling and analyzing tools Support for automated testing frameworks	Resource-intensive Requiring high system requirements Requires prior programming knowledge The size of the app built on Android Studio can be larger compared to other platforms. Development and deployment of apps on Android Studio can be time-consuming Requiring considerable testing and optimization efforts Compatibility issues with some older devices
Xamarin	Write code in C# then compile it to the native language of any platform Provide access to native APIs and libraries for each platform Code Sharing between iOS, Android, and Windows apps Have pre-built UI Controls that can create native user interfaces quickly and easily. Integration with Visual Studio	Steep Learning Curve (requires knowledge of C# and native development environment) Limited Access to Latest Features because it relies on its own version Increased App Size because Xamarin requires additional native APIs Limited Community Support as compared with Android and IOS
Xcode	Integration with Apple Platforms Has robust support for Swift Programming Language Interface Builder allows developers to design interfaces by drag-and-drop Comprehensive Toolset which includes a source code editor, debugging tools, performance analyzers, asset management, version control integration, and testing frameworks Simulator and Emulator App Store Deployment Extensive Documentation and Community Support	Limited Platform Support not for Android or Windows Resource Intensive for large projects or using features like Interface Builder Steep Learning Curve(complex and extensive set of features for new developers) Interface Builder has Limitations in terms of flexibility and customization (manually written code) Code Completion and Indexing Issues in large projects Xcode IDE may occasionally experience stability issues or crashes.

Unity	Cross-Platform Development Visual Editor and Workflow offers drag-and-drop and+D68 visual scripting Asset Store and Community Strong 2D and 3D Capabilities Scripting and customization provide access to Unity's extensive API Provide support to performance and optimization tool Unity Collaborate allows teams to work together.	Performance Challenges because general-purpose design may result in overhead Steep Learning Curve because understanding the various components may take time Scripting Limitations because specific programming paradigms may be more challenging to implement Documentation and Support (some parts of the documentation may be outdated, incomplete, or lack detailed explanations) Asset Store Quality Control because some assets may lack documentation or have compatibility issues Large Build Sizes Software Updates(updating Unity to a new version can sometimes introduce issues)

Table 1: Provide information about the strengths and weaknesses of the development platforms.

In Table 2, we have mentioned the names of the platforms, types (open-source, not open source or offers free and paid licensing), names of the plugins and extensions, best-suited information (i.e., Native: Android and iOS, Cross-platform: iOS, Android, and Windows) and finally listed the accessibility features.

Platforms Name	**Type**	**Plugins and extensions**	**Best suited for**	**Accessibility features**
Android Studio	Open-source	Firebase Android ButterKnife Zelezny Android Material Design Icon Generator Genymotion Android WiFi ADB CodeGlance Android Lint	Native Android apps	Screen reader compatibility Text-to-speech and speech-to-text conversion Closed captioning High contrast mode Adjustable font size Customizable color schemes Voice recognition
Xamarin	Open-source	Xamarin.Forms Xamarin.Essentials Prism Syncfusion HockeyApp	Cross-platform development iOS, Android, and Windows. Native mobile apps with a shared codebase.	Support for Accessibility APIs Customizable User Interfaces Focus Navigation Accessibility Testing Accessibility Guidelines
Xcode	Not open source	Alcatraz Injection for Xcode SwiftLint ClangFormat XcodeGen KSImageNamed	Apple's platforms, including iOS, macOS, watchOS, and tvOS	VoiceOver Support Keyboard Navigation Voice Control Dynamic Type Support Color Vision Accessibility Accessibility Inspector

		Dash for Xcode GitHub for Xcode Reveal		Documentation Accessibility
Unity	Not open source offers both free and paid licensing.	Rider MonoDevelop-Unity ProBuilder Playmaker Odin Inspector VFX Graph	Cross-platform game development and interactive applications	Text-to-Speech (TTS) Support Screen Reader Compatibility Subtitle Support High Contrast Mode Keyboard and Gamepad Remapping UI Scaling and Resolution Independence

Table 2: Provides information about the type, plugins, extensions, best suitable for and accessibility features of the development platforms.

Table 3 summarizes information about the accessibility API, tools packages and their features (i.e., 1=Event Monitoring, 2=UI Navigation, 3=Content Descriptions, 4= Gesture Detection, 5= Text-to-Speech/Voice Over Support, 6= Event Filtering, 7= Accessibility Settings, 8= Hints and Lables, 9= Screen Reader and 10= Subtitles and Closed Captions).

Name	**Accessibility APIs, tools and Packages**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Xamarin	AccessibilityDelegate AccessibilityService TalkBack Support Xamarin.Forms Accessibility API Xamarin.Android Accessibility Xamarin.Essentials Xamarin.iOS Accessibility APIs Custom Accessibility Implementations	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
Android Studio	AccessibilityService AccessibilityNodeInfo AccessibilityEvent AccessibilityManager ExploreByTouchHelper AccessibilityDelegate AccessibilityAction AccessibilityNodeProvider	Y	Y	Y	Y	Y	Y	Y	Y	Y	N
Xcode	UIAccessibility UIAccessibilityElement UIAccessibilityContainer UIAccessibilityCustomAction UIAccessibilityFocus UIAccessibilityIdentification	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

	UIAccessibilityTraits UIAccessibilityZoom NSAccessibility										
Unity	Unity's Accessibility Input System Text-to-Speech UIElements EventSystem Unity's UGUI Accessibility Inspector Unity Accessibility Package	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

Table 3: Provides information about the accessibility API, tools packages and their features.

3 NATIVE and Cross-platform screen reader APIs

The functionality and capabilities of the screen reader software on a particular platform are directly accessible through native screen reader APIs. These APIs allow programmers to interact with the screen reader, get data from the screen, and create accessible information for their apps. We thoroughly reviewed the screen reader APIs, taking into account our past knowledge as well as the iOS and Android programming development manuals and documentation. Based on the functionality exposed, we developed an API taxonomy. We have listed some API functionalities and their compatibility with the four platforms under consideration (native iOS, native Android, Unity, and Xamarin). APIs that support fundamental screen reader features are also shown in Table 4. We used the accessibility guidelines and best practices listed in Apple and Google's introductory accessibility documentation [13-16] to identify fundamental features.

3.1 Android Studio Accessibility APIs

AccessibilityService and AccessibilityNodeInfo are important accessibility APIs of Android Studio. AccessibilityService API allows programmers to develop services that interact directly with the screen reader to process the accessibility events. On the other hand, AccessibilityNodeInfo will enable developers to adjust accessibility information for their Android applications by providing information about the characteristics and state of UI elements.

3.2 iOS Accessibility APIs

UIAccessibility, UIAccessibilityElement and NSAccessibility are important accessibility APIs of XCode. UIAccessibility offers a collection of classes and protocols for improving iOS application accessibility, including VoiceOver support. These APIs allow developers to give UI elements accessibility labels, traits, hints, and other data. UIAccessibilityElement enables the creation of custom accessibility components and sharing this information through VoiceOver in iOS applications. NSAccessibility provides a complete set of classes and protocols for adding accessibility features to macOS apps. These APIs allow developers to handle accessibility events, handle the change in accessibility events and provide accessibility information on these events.

3.3 Xamarin Accessibility APIs

AccessibilityService, UIAccessibility and Xamarin.Forms are essential accessibility APIs of Xamarin. AccessibilityService enables programmers to build services that can handle accessibility

events. With the help of this API, you may communicate with the screen reader for Android and can customize the UI elements. UIAccessibility offers a collection of classes and protocols for improving the accessibility of iOS applications. Developers can make UI elements more accessible with the VoiceOver screen reader by using these APIs to offer accessibility labels, traits, hints, and other information. Xamarin.Forms ensure accessibility support using the native accessibility APIs on each platform. Developers can use Xamarin to provide accessibility information, manage accessibility events, and create Xamarin forms with accessibility features and characteristics programs that use screen reader-friendly forms.

3.4 Unity Accessibility APIs

Unity offers no particular screen reader API. On the other hand, Unity provides some features that allow developers to integrate accessibility features into their Unity apps. The accessibility features can be implemented by making them compatible with screen readers on several platforms. Here are some strategies for integrating accessibility for screen readers in Unity: Text-to-Speech (TTS) Integration, Custom Accessibility Implementation and User Interface (UI) Navigation and Interaction. Table 4 indicates which APIs implement basic screen reader functionalities.

We considered the APIs mentioned above and organized them into the following five categories:

Accessibility focus: The ability for the user to select a view, hear a description of it, and then optionally activate it, is a fundamental way of mobile screen reader interaction. Once a view is selected, we can say this is currently in accessibility focus; only one view may be given the focus at once.

Text-to-speech: The screen reader speaks aloud the accompanying text description when a view is under the accessibility focus. The developers can define this description, which we refer to as text-to-speech.

Explicit TTS: Text-to-speech (TTS) software reads the text and speaks aloud. The programmers can also utilize TTS to read aloud content automatically.

Accessibility Hierarchy: The logical hierarchy structure defines the screen elements to enhance the accessibility of the screen contents. The developer has access to change this structure.

Other: Any other accessible features

ID	Category	API Functionality	XCode	Android	Xamarin	Unity
1	Accessibility Focus	Indicate which view received the accessibility focus	Y	Y	Y	Y
2		State the accessibility focus order.	Y	Y	Y	Y
3		Indicate action connected to accessibility focus-related events (Like toggle view focus)	Y	Y	N	N
4		Determine which view has the accessibility focus	Y	Y	N	Y
5	Text-to-speech	Indicate the position where TTS perform	Y	Y	L	L
6		Mention TTS with the help of a program	N	Y	N	Y
7		One view defines another view.	N	Y	N	N
8		Notice the change even without	N	Y	N	Y

		user interaction				
9	Explicit TTS	Inform when the screen reader finishes reading	Y	N	N	Y
10		Can change TTS features i.e., speech or pitch	Y	Y	N	N
11		Read a text with a non-screen reader TTS	Y	Y	Y	Y
12		Identify when reading performed by a non-screen reader	Y	Y	N	N
13		non-screen reader can change TTS features i.e., speech or pitch	Y	Y	Y	Y
14	Accessibility Hierarchy	Combine various views into a single element that is easily accessible.	N	Y	Y	Y
15		Divide a view into various accessibility components.	Y	Y	N	Y
16		Can access the parent element from the child	Y	Y	N	Y
17	Other	Detection of an active screen reader	Y	Y	N	N
18		Provide implementation on how to respond to a screen reader action	Y	N	Y	Y
19		Take action on the user's behalf.	N	Y	N	N
20		Get arbitrary elements when accessibility-related data you want to view.	N	Y	N	N

Table 4 indicates which APIs implement basic screen reader functionalities.

4 Discussion and Conclusion

The decision to develop Android, iOS, Xamarin, or Unity apps depends on several factors, including the project's requirements, the intended audience, and the developer's experience. Regarding accessibility features and screen reader APIs, every platform has its own advantages and disadvantages. Android can be fragmented but offers diversity and open-source freedom. While limiting customization, iOS promotes consistency and accessibility. Xamarin and Unity offer cross-platform development solutions, with Xamarin giving C# compatibility and Unity being open-source. Both support accessibility and have access to native functionality.

These platforms offer unique approaches to app development, which we have characterized by their strengths, weaknesses, and accessibility features. Android, an open-source platform, boasts many hardware options and incorporates inclusive features such as TalkBack and magnification gestures. Nevertheless, it contends with device fragmentation and potential delays in updates, which may affect the uniformity of the user experience. On the other hand, iOS is renowned for its accessibility-centric approach, delivering a tightly integrated ecosystem complete with VoiceOver and consistent updates,

ensuring high accessibility support. However, its closed nature may limit customization options, with the result that apps developed using these tools may present a steep learning curve for users.
Xamarin and Unity, serving as cross-platform development frameworks, provide advantages like code reusability and access to native features, enhancing efficiency. Xamarin, with its compatibility with C# and platform-specific accessibility APIs, empowers developers to create accessible apps. Developers require additional efforts to get familiar with the different platform-specific features to implement the Native APIs. On the other hand, Unity, being open-source, can harness native accessibility features but may encounter delays in adopting new platform features and require developers to acquaint themselves with its framework. Irrespective of the chosen platform or framework, placing accessibility at the forefront of app development is imperative to ensure that all users, including those with disabilities, can fully utilize and benefit from the created applications. Developers should conscientiously evaluate these factors in light of their project prerequisites and available resources when selecting.
Furthermore, native APIs provide a robust and deeply integrated accessibility solution for Android and iOS. Cross-platform solutions like Xamarin and Unity offer the benefit of code reusability and the potential to tap into native features. However, developers must be mindful of potential delays in adopting new platform features and the learning curve associated with cross-platform frameworks. In both platforms, the paramount consideration is accessibility in app development. This ensures that all users, including those with disabilities, can access and utilize apps effectively. Developers should select the best approach with their project requirements, user needs and available resources while keeping accessibility at the forefront during the development of the apps.

Acknowledgment

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant number 18/CRT/6222. For the purpose of Open Access, the author has applied a CC BY public copyright licence to any Author Accepted Manuscript version arising from this submission.

References

1. The Rise of Android: Why is Android Successful? (Source: https://www.bankmycell.com/blog/how-many-android-users-are-there)
https://www.bankmycell.com/blog/how-many-android-users-are-there#:~:text=Android%20Users%3A%20Today%2C%20there%20are,mobile%20operating%20system%20market%20share.
2. Number of apps available in leading app stores as of 3rd quarter 2022 https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/
3. Lecomte, S., & Martinez, M. (2017). Towards the Quality Improvement of Cross-Platform Mobile Applications. In IEEE/ACM 4th International Conference on Mobile Software Engineering and Systems (MOBILESoft) (pp. pp. 184-188).
4. Toasa, R. M., Egas, P. F. B., Recalde, H., & Saltos, M. G. (2023, February). Mobile Development with Xamarin: Brief Literature, Visualizations and Important Issues. In International Conference on Information Technology & Systems (pp. 299-307). Cham: Springer International Publishing.
5. Bin Uzayr, S. (2022). Mastering Unity: A Beginner's Guide. CRC Press.
6. Balaji, V., & Kuppusamy, K. S. (2016, December). Accessibility analysis of e-governance oriented mobile applications. In 2016 International Conference on Accessibility to Digital World (ICADW) (pp. 141-144). IEEE.

7. Acosta-Vargas, P., Salvador-Acosta, B., Salvador-Ullauri, L., Villegas-Ch, W., & Gonzalez, M. (2021). Accessibility in native mobile applications for users with disabilities: A scoping review. Applied Sciences, 11(12), 5707.
8. Peter Korn, Evangelos Bekiaris, and Maria Gemou. 2009. Towards open access accessibility everywhere: The ÆGIS concept. In International Conference on Universal Access in Human-Computer Interaction. Springer, 535–543
9. Christoph Rieger, Daniel Lucrédio, Renata Pontin M Fortes, Herbert Kuchen, Felipe Dias, and Lianna Duarte. 2020. A model-driven approach to cross-platform development of accessible business apps. In Proceedings of the 35th Annual ACM Symposium on Applied Computing. 984–993
10. Disability https://www.who.int/en/news-room/fact-sheets/detail/disability-and-health
11. Charland, A., & Leroux, B. (2011). Mobile application development: web vs. native. Communications of the ACM, 54(5), 49-53.
12. Xanthopoulos, S., & Xinogalos, S. (2013). A comparative analysis of cross-platform development approaches for mobile applications. BCI '13, (pp. 213-220).
13. Apple. 2023. Apple Human Interface Guidelines. Accessibility section. https://developer.apple.com/design/human-interface-guidelines/accessibility
14. Google. 2023. Android Developers Documentation. Principles for improving app accessibility. https://developer.android.com/guide/topics/ui/accessibility/principles
15. Niccolò Cantù, Mattia Ducci, Dragan Ahmetovic, Cristian Bernareggi, and Sergio Mascetti. 2018. MathMelodies 2: a Mobile Assistive Application for People with Visual Impairments Developed with React Native. In ACM SIGACCESS Conference on Computers and Accessibility (ASSETS). ACM.
16. Andres Gonzalez and Loretta Guarino Reid. 2005. Platform-independent accessibility api: Accessible document object model. In Proceedings of the 2005 International Cross-Disciplinary Workshop on Web Accessibility (W4A). 63–71.