Comparing App Development Approaches

Native vs. Cross-Platform App Development

Katharina Wurm
it241504@fhstp.ac.at

05.03.2025

ABSTRACT

The increase in mobile device usage has driven the need for applications that work across various platforms. This paper explores the two primary approaches to mobile app development: native and cross-platform. Existing research shows that native development ensures high performance and user experience but requires more time and resources. In contrast, cross-platform development methods – such as web, hybrid, interpreted, and cross-compiled apps – streamline development through code reuse, though often at the cost of performance and functionality. However, as mobile technologies continue to evolve, research gaps remain, thus emphasising the need for updated studies to reflect the rapid technological advancements in this field.

KEYWORDS

Mobile, App development, Native app development, Cross-platform app development

1 Introduction

Nowadays, mobile phones have become a device that most people use every day, and the various types of smartphones on the market are almost as diverse as the people using them. Thus, developers intending to create an app need to make sure that it can run on different operating systems. By making an app available on as many mobile platforms as possible, the app's market presence can be increased, which according to Delía et al. (2017) is vital, as the success of an app is tied to its popularity.

However, due to the numerous possibilities regarding programming languages, development tools, time-to-market demands, and more, the choice on how to make the app work on multiple platforms is not always easy (Delía et al., 2017), especially also because of the ever-changing and constantly evolving technologies (Karami et al., 2023).

There are two different approaches to app development, which will both be discussed in this paper: native and cross-platform development. Both of them have their advantages and disadvantages, which all need to be considered when choosing between the available app development frameworks (Karami et al., 2023). The goal of this paper is to examine and compare other researchers' findings on app development approaches and to identify gaps that need to be addressed through future research.

2 Types of App Development

2.1 Native App Development

When developing mobile applications natively, a development project is created and worked on for each platform, which uses the platform-specific languages (Delía et al., 2017), such as Java or Kotlin for Android and Swift for iOS (Koram & Garg, 2023; Suri et al., 2023). Thus, when wanting to distribute an app to different platforms, the same app has to be developed multiple times (Koram & Garg, 2023). Since a separate app needs to be developed for each platform, the development and maintenance of apps costs more and demands more effort (Delía et al., 2015); due to little code reuse, it is very expensive and time-consuming (Suri et al., 2023).

However, these extra costs also have their advantages: native app development allows developers to directly access native platform functionalities, such as the camera, GPS, and other sensors (Delía et al., 2017; Ebone et al., 2018; Koram & Garg, 2023). Furthermore, according to Delía et al. (2017), a user does not need to be connected to the internet to be able to run a native application. Additionally, the researchers claim that native apps execute quickly and it is possible for them to run in the background. Generally, Ebone et al. (2018) believe that native apps provide the best performance and user experience on their respective platforms. Patidar and Suman (2021) are of a similar opinion, since they state that native apps tend to allow for high performance and good user experiences. Koram and Garg (2023) also highlight the "significantly better performance" of native apps and their secure and robust user interfaces (p. 262). Because of their performance and fast execution time, native apps typically have a higher ranking in app stores, meaning they have a "significantly higher level of user satisfaction" (Suri et al., 2023, p. 2).

When it comes to native applications in the academic field, Karami et al. (2023) evaluated 75 peer-reviewed conference and journal papers on different app development approaches and noticed that Android and iOS were the first and fifth most studied frameworks. The researchers believe this might be because half of the evaluated studies used a native app development framework as a baseline to compare cross-platform frameworks to.

2.2 Cross-Platform App Development

Cross-platform, or multi-platform, development uses a single code base which can be run on different mobile platforms (Delía et al., 2017; Ebone et al., 2018; Suri et al., 2023). Thus, the advantage of this development approach is the reduction of development effort, cost and time (Delía et al., 2017; Ebone et al., 2018; Suri et al., 2023) due to the code reutilisation (Delía et al., 2015). However, cross-platform apps are unable to provide the same performance and user experience as native applications (Suri et al., 2023).

According to Karami et al. (2023), the academic interest in cross-platform frameworks is advancing: Cordova appeared within 32 studies and is therefore the most studied multi-platform framework. However, Cordova, as well as Titanium, are also among the oldest cross-platform frameworks, as they were released in 2009. The newer frameworks include Xamarin, which was released in 2011, React Native, which came out in 2016, and Flutter, which was released a year later. In 2021, Flutter was very popular among developers (Koram & Garg, 2023). Karami et al. (2023) also found Flutter and React Native to have been the most popular multi-platform frameworks among developers in 2022, thus beating Ionic and Xamarin.

Although the mentioned frameworks are all cross-platform, they can differ in how they work. There are multiple approaches to cross-platform app development, which will be elaborated on in the following subchapters.

2.2.1 Web Applications

Web applications are apps that are developed with HTML, CSS and JavaScript, and run in the browser (Delía et al., 2017; Koram & Garg, 2023). No installation is necessary, which means the distribution and updating of apps is easier (Delía et al., 2017; Koram & Garg, 2023), but it might make the application less attractive compared to native apps (Delía et al., 2015). Since only a browser with a connection to the internet is needed, the development is fast and easy, and the developer does not need to adapt to any specific platform (Delía et al., 2017).

On the downside, due to the client-server interaction via the internet, the response time and performance of the web app could be affected negatively (Delía et al., 2015), which could worsen the user experience (Delía et al., 2017). In addition to web apps being dependent on an internet connection most of the time, the type of browser may also affect the user experience (Koram & Garg, 2023). Furthermore, because of the security restrictions imposed on websites, web apps and their interfaces are limited when it comes to the use of native functionalities (Delía et al., 2015; Delía et al., 2017).

Nowadays, Progressive Web Apps (PWAs) aim to make web apps feel more like real apps, by even making them able to be run offline (LePage & Richard, 2024), which is why it would be interesting to further investigate how modern PWAs compare in terms of performance and usability.

2.2.2 Hybrid Applications

Hybrid applications rely on the use of web technologies, such as HTML, CSS and JavaScript, but unlike web apps, they are not run by a browser; instead, they are executed on a web container, which includes an API that grants access to device-specific functionality (Delía et al., 2017). This way, according to Delía et al. (2017), code can be reused for different platforms, all while still being able to distribute the app via the platform-specific app stores and being able to access native functionalities of the device. Thus, unlike web apps, hybrid apps can function offline and are "true applications" (Koram & Garg, 2023, p. 263).

On the other hand, the web container may negatively impact the performance of the app, as it requires an extra load (Delía et al., 2017; Koram & Garg, 2023; Zou & Darus, 2024). Moreover, since it is developed using web technologies, the lack of native components in the user interface might have a negative effect on the user experience (Delía et al., 2017).

Examples of hybrid development frameworks include Apache Cordova (Delía et al., 2017) and Ionic (Zou & Darus, 2024). The latter also allows apps to be run directly in a web browser as a PWA in addition to distributing the app via native app stores (Pinto & Coutinho, 2018).

2.2.3 Interpreted Applications

Interpreted applications are often also built with JavaScript, but the majority of the project is translated to native code, with the remainder being interpreted at runtime, so that native interfaces are obtained (Delía et al., 2017). Due to the native components, a high performance can be achieved (Zou & Darus, 2024).

Some examples of frameworks that produce interpreted applications include Appcelerator Titanium and NativeScript, according to Delía et al. (2017). The researchers also state that Titanium's API serves as a bridge, since it maps each JavaScript element to its matching native element, thus offering natively controlled user interfaces. Something similar is reported by Zou and Darus (2024): React Native allows developers to build performance-critical portions of the application in native languages due to its bridge which connects JavaScript with native modules. The fact that lots of components are directly translated to their native equivalents allows for responsive UIs and high performance; nevertheless, the JavaScript bridge may pose as a bottleneck at times (Zou & Darus, 2024).

2.2.4 Applications Generated by Cross-Compilation

Some applications can be directly compiled into native code, such as by using the Xamarin or Corona framework, so that an app version for each target platform is generated (Delía et al., 2017).

However, while in Corona, only one base code is written in Lua, a simple scripting language, Xamarin, on the other hand, only allows developers to write shared business logic code in C#, and each platform's user interface still must be developed separately (Delía et al., 2017). But then again, Zou and Darus (2024) note that even though platform-specific user interfaces can be created using Xamarin.iOS and Xamarin.Android, developers can also choose to build a shared UI with Xamarin.Forms. Delía et al. (2017) might not have mentioned Xamarin.Forms because it was less widely adopted or fully developed at the time of their publication. Nevertheless, even though Xamarin.Forms allows

developers to build a shared user interface, it often comes with trade-offs in terms of delivering a fully native user experience, as Zou and Darus (2024) emphasise.

Still, the approach of generating apps by cross-compilation often yields a performance close to that of native apps, and results in apps that look and behave similarly to as if they were written natively (Ebone et al., 2018; Zou & Darus, 2024). Furthermore, by compiling directly to native code, a framework like Flutter can get rid of issues associated with JavaScript bridges in other frameworks (Zou & Darus, 2024). However, Flutter requires developers to use a specialised language called Dart, which they must learn in order to build Flutter applications (Suri et al., 2023).

2.3 Summary

To sum up, apps can be developed natively or by using the cross-platform approach. There are several types of cross-platform development which can be further categorised into web apps, hybrid apps, interpreted apps, and apps generated by cross-compilation. Each of these approaches differs, so depending on the specific requirements of an app, some might be better suited than others. Therefore, it is important to understand how these types of app development compare across different metrics, which will be elaborated on in the following chapter.

3 Comparison

When it comes to comparing different frameworks or types of app development, both the user's perspective – such as the end user's satisfaction with the app's performance and user interface – and the developer's perspective – such as the amount of support provided by the framework during development – can be considered (Karami et al., 2023). Out of all the studies assessed by Karami et al. (2023), almost half of them took both perspectives into account, while the other half was split relatively evenly between the two. Furthermore, the researchers highlighted that the criteria used to analyse the user's perspective on an app's performance and UI often included CPU usage, memory, battery level, launch time, and frames per second, while the criteria used to evaluate the developer's perspective varied; examples included access to device sensors and the availability of framework documentation and support. On average, each study focussed on four criteria to compare different app development approaches.

The most used criterion for comparison is performance, as it was found by Karami et al. (2023) in more than half of the evaluated studies. According to Delía et al. (2017), this is because performance highly influences user experience, and a bad user experience will result in unhappy users, and thus bad user ratings. Furthermore, Karami et al. (2023) found the next most used criteria to be, in order: platform API accessibility, hardware and sensors accessibility and user interface, so the app's interface's quality from the user's point of view. Thus, the main criteria include both some related to the user's perspective, and some related to the developer's perspective.

Additionally, Karami et al. (2023) categorised studies based on their evaluation methods: experiment-based studies assessed frameworks through prototype testing, while documentation-based studies analysed the frameworks' documentation. The researchers found that two-thirds of the studies were experiment-based, providing insights into runtime performance but with results limited to specific prototypes – in contrast, document-based studies offered a broader API evaluation but lacked real-world performance insights. Few studies combined both methods or included user surveys, likely due to the time and effort required (Karami et al., 2023).

The following subchapters summarise findings by various researchers on how different app development approaches compare in specific metrics.

3.1 Performance

Karami et al. (2023) found that when it comes to performance, native frameworks always yield a better result than multi-platform frameworks. Furthermore, the researchers discovered that React Native was often perceived to affect performance negatively in comparison to other cross-platform frameworks.

On the other hand, Ebone et al. (2018) did not notice a relevant difference between native Android and iOS apps, a Xamarin Android app, and Appcelerator Titanium Android and iOS apps, while the Xamarin.Forms Android and iOS apps took significantly longer to load larger views. On top of that, the researchers detected that the UI response time patterns of the Apache Cordova apps on Android and iOS varied greatly on the platforms.

Similarly, Delía et al. (2017) also highlighted a difference between the Android and iOS platforms; they concluded that performance experiments involving the iOS and Android operating systems ought to be examined independently, as the native method in iOS is far more effective. The researchers believe that the native approach in Android might be slowed down because Java needs the Android Runtime (ART) to function. On the iOS platform, the native app also showed a significantly higher performance compared to the cross-compiled apps (Delía et al., 2017). Nevertheless, regardless of the operating system, a web development approach would be an easy way to get a high performance on all mobile devices, but only if the access to native functionalities of the device is not required (Delía et al., 2017). When analysing hybrid (Cordova) and interpreted apps (Titanium and NativeScript), Delía et al. (2017) realised that the type of JavaScript engine used in both approaches had a big influence on the apps' performance: hybrid and interpreted apps running on Android, which uses the JavaScript V8 engine, had a much better performance than those running on iOS, which uses the JavaScriptCore engine. Hybrid and interpreted apps running on Android showed a similar performance to web apps and were thus even better than the native and cross-compiled apps, while the same apps running on iOS were worse than native, web and cross-compilation approaches (Delía et al., 2017). Thus, according to Delía et al. (2017), the apps which were cross compiled using Xamarin and Corona had the worst performance on the Android platform, but only the second worst on iOS. However, it is important to acknowledge that this study from 2017 may not fully

reflect the performance of modern app development frameworks, thus highlighting the need for ongoing research in this field.

In a more recent paper, Koram and Garg (2023) stated that Flutter "is known for its high level of performance" (p. 264). Furthermore, the researchers found Ionic to have a moderate performance and React Native and Xamarin to have a performance "quite similar to native" (p. 265). Therefore, it would be interesting to analyse how different app development approaches compare in terms of performance today.

3.2 Memory, Battery, and CPU Usage

When it comes specifically to memory, native apps showed to use the least compared to Flutter and React Native apps, with the latter using the most (Karami et al., 2023). Furthermore, Karami et al. (2023) found React Native to also consume more CPU and battery than Flutter. Similarly, Suri et al. (2023) found React Native to have the highest memory and CPU usage compared to Kotlin and Flutter, but when it came to energy usage, both React Native and Flutter showed a very high usage. Given these findings, it would be valuable to examine how other frameworks than the ones mentioned compare in terms of memory, CPU, and battery consumption.

3.3 App Size

Compared to native Android and iOS apps, cross-platform apps are substantially larger (Ebone et al., 2018). Karami et al. (2023) point out that React Native apps are especially large. A similar finding was reported by Suri et al. (2023), who found that when comparing an app made with Kotlin, Flutter and React Native, the native Kotlin app used the least amount of space, while the React Native app used the most. This might also explain the before-mentioned high memory, battery and CPU usage by React Native.

3.4 Platform API and Hardware and Sensors Accessibility

Moving on, when it comes to the access to camera, geolocation, notifications, etc., most studies observed native frameworks to be better, especially since most multi-platform frameworks rely on third-party libraries to access some APIs (Karami et al., 2023). This finding is in line with the frameworks' descriptions in Chapter 2.

3.5 Development Support

While relying on third-party libraries can sometimes pose limitations, it is also a significant advantage. According to Karami et al. (2023), popular frameworks like React, Ionic, and Flutter offer more third-party libraries and plugins and have greater development support. However, it is not clear whether this statement only applies in comparison to other cross-platform frameworks, or also in comparison to native app development methods.

3.6 Code Smells and Bugs

Another metric of interest is the number of code smells and bugs. Karami et al. (2023) define a code smell as a "maintainability issue that makes your code confusing and difficult to maintain" (p. 136). When comparing native Android apps with React Native apps, the researchers noticed that native Android apps have more code smells. However, in large apps, native Android apps still tend to have less bugs than React Native apps. Based on these findings, it would be interesting to assess even more frameworks regarding their number of code smells and bugs.

3.7 User Interface

Lastly, when it comes to the user interface criterion, there is no clear winner on the cross-platform side; however, generally, most studies believe native frameworks to be more efficient in producing a high-quality interface than cross-platform ones (Karami et al., 2023). Which frameworks were evaluated in specific, though, is not clear.

Furthermore, Koram and Garg (2023) also believe native apps to be the preferred choice for delivering an intuitive user interface due to their high performance and reliability.

4 Conclusion

In the matter of app development, there are various approaches one can take. Choosing between them can be difficult, especially since each one of them has its advantages and disadvantages.

Patidar and Suman (2021) believe that developers should choose the native approach if they need the best user experience, the fastest speed, the possibility to fully make use of device features, and if they need the app to function even without an internet connection. Moreover, Karami et al. (2023) speculate that complex applications, such as banking apps, may need to be developed natively to ensure a better UI.

However, the native approach is expensive and takes a lot of time, so developers might choose to go for a cross-platform development approach. The easiest way to reach as many users as possible is to create a web app, as all that is required for distribution is a browser. Nevertheless, a web app is limited when it comes to accessing native device functionality. Another alternative is to develop a hybrid app, which according to Delía et al. (2015) still keeps the development effort small, similarly to the web approach. Furthermore, interpreted apps and cross-compiled apps are among the options as well, which are the most appropriate when the developer values good performance and user experience (Delía et al., 2015), but still wants to save time and money.

Whether or not a framework is a good choice might also look different from the perspective of a user compared to the one from a developer. For example, Karami et al. (2023) identified that Flutter is better from a performance standpoint, so it might improve user experience, while React Native is better in the case of development support. Thus, whether an approach or framework

is the right choice depends on various factors, and it is up to the developer to decide which aspects to prioritise.

4.1 Limitations and Future Work

Due to the ever-evolving nature of technology, it must be considered that some of the papers cited in this work were published several years ago, and their findings may not fully align with the current state of technology anymore. For example, the tests in the study by Delía et al. (2017) were done on devices like the Samsung S6 and the iPhone 6 plus, both which were considered high-end at the time, but would not anymore in 2025. This highlights the need for new studies to validate, refine, or challenge these earlier findings. Conducting research with the frameworks and devices available today can provide updated insights and ensure that our understanding keeps up with current developments.

REFERENCES

Delía, L., Galdamez, N., Thomas, P., Corbalan, L., & Pesado, P. (2015). Multi-platform mobile application development analysis. *2015 IEEE 9th International Conference on Research Challenges in Information Science (RCIS)*, 181–186. https://doi.org/10.1109/RCIS.2015.7128878

Delía, L., Galdamez, N., Corbalan, L., Pesado, P., & Thomas, P. (2017). Approaches to mobile application development: Comparative performance analysis. *2017 Computing Conference*, 652–659. https://doi.org/10.1109/SAI.2017.8252165

Ebone, A., Tan, Y., & Jia, X. (2018). A Performance Evaluation of Cross-Platform Mobile Application Development Approaches. *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, 92–93. https://ieeexplore.ieee.org/document/8543442

Karami, P., Darif, I., Politowski, C., El Boussaidi, G., Kpodjedo, S., & Benzarti, I. (2023). On the Impact of Development Frameworks on Mobile Apps. *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, 131–140. https://doi.org/10.1109/APSEC60848.2023.00023

Koram, N., & Garg, R. (2023). Review on Mobile App Development: Tools and Techniques. *2023 IEEE World Conference on Applied Intelligence and Computing (AIC)*, 260–266. https://doi.org/10.1109/AIC57670.2023.10263908

LePage, P., & Richard, S. (2024, September 19). *What makes a good Progressive Web App?* web.dev. https://web.dev/articles/pwa-checklist

Patidar, A., & Suman, U. (2021). Towards Analyzing Mobile App Characteristics for Mobile Software Development. *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*, 786–790. https://ieeexplore.ieee.org/document/9441097

Pinto, C. M., & Coutinho, C. (2018). From Native to Cross-platform Hybrid Development. *2018 International Conference on Intelligent Systems (IS),* 669–676. https://doi.org/10.1109/IS.2018.8710545

Suri, B., Taneja, S., Bhanot, I., Sharma, H., & Raj, A. (2023). Cross-Platform Empirical Analysis of Mobile Application Development frameworks: Kotlin, React Native and Flutter. *ICIMMI '22: Proceedings of the 4th International Conference on Information Management & Machine Intelligence*, 1–6. https://doi.org/10.1145/3590837.3590897

Zou, D., & Darus, M. Y. (2024). A Comparative Analysis of Cross-Platform Mobile Development Frameworks. *2024 IEEE 6th Symposium on Computers & Informatics (ISCI)*, 84–90. https://doi.org/10.1109/ISCI62787.2024.10667693