Proceedings of the Pakistan Academy of Sciences:
A: Physical and Computational Sciences 58(1): 35–45 (2021)
Copyright © Pakistan Academy of Sciences
ISSN (print) 2518-4245; ISSN (Online) 2518-4253
https://doi.org/10.53560/PPASA(58-1)717

Pakistan Academy of Sciences

Research Article

A Comparative Analysis of Mobile Application Development Approaches

Mohamed Abdal Mohsin Masaad Alsaid[1], Tarig Mohamed Ahmed[1,3], Sadeeq Jan[2*] , Fazal Qudus Khan[3], Mohammad[2], and Amjad Ullah Khattak[4]

[1]Department of Computer Science, University of Khartoum, Khartoum, Sudan
[2]Department of Computer Science & IT, University of Engineering and Technology, Peshawar, Pakistan
[3]Department of Information Technology, King Abdulaziz University, Jeddah, Saudi Arabia
[4]Department of Electrical Engineering, University of Engineering and Technology, Peshawar, Pakistan

Abstract: Over the last decade, there has been a significant increase in the development of mobile applications. The performance of the developed applications depends largely on the development approaches. There are two widely used approaches: (1) native, where the application is targeted and developed for a specific platform, (2) cross-platform, where the developed application runs on multiple platforms. This paper aims to address the question of which approach should be used in various scenarios. We have performed a detailed comparison of the two approaches by developing a mobile app using both approaches. Experiments are performed using Android and iOS, the two most well-known mobile Operating System. The criteria of deciding the best approach include performance, usability and support. Our results show that both approaches are viable depending on the requirements and type of the application to be developed, with native having an edge. Guidelines are presented at the end to help the developers in choosing the best approach. The fundamental differences and advantages of each approach are discussed.

Keywords: Mobile Applications, Native, Cross-platform, Development, Programming.

1. INTRODUCTION

In the age of technology, smartphones are the most widely used electronic device, used by billions of people. They have become a key part of our daily lives. But what makes a smart-phone "smart" is the functionality it provides beyond phone calls and texting, such as touch screens, GPS, camera, biometrics, and that they have fully capable operating systems that can run all sort of applications, like email, browsers, banking, health care, games and much more. There exist several smart-phone platforms, however, Android and IOS have been dominant, with their market share being 87.7% and 12.1%, respectively [1].

Android is an Open source mobile operating system that is developed and maintained by Google and is used by many smart-phones manufactures. IOS is a closed source mobile operating system developed by Apple company for their own devices. In the process of making a mobile application, one of the first decisions to make is choosing the target platforms and the technology stack to use for development. For most cases targeting Android and IOS is the way to go. The ideal situation would be that both platforms can run the same source code without compromising performance, support or usability.

With two major operating systems, it makes sense to target both platforms when developing an app that needs to target almost all users. There are two General approaches in mobile application development, native and cross-platform. Native application development involves using the

Received: July 2020; Accepted: March 2021
*Corresponding Author: Sadeeq Jan <sadeeqjan@uetpeshawar.edu.pk>

supported languages and APIs directly, Native development in Android is done with Java/Kotlin and in ISO with Objective-C/Swift. Cross-platform solutions are made to achieve the goal of "write once run everywhere", the same source code runs in multiple platforms without the need to use platform-specific code (at least for most cases) and is usually done by implementing an abstraction layer upon wish the cross-platform code runs. Some notable cross-platform mobile solutions are React-native, Unity, Flutter, Apache Cordova, Xamarin and Kotlin-native.

The native application development approach adds to the performance of the mobile apps; however, it comes with additional technical and financial costs during the development maintenance phase. On the other hand, the cross-platform approach is beneficial for the users, however, it has several limitations like inferior performance and lack of support etc. Deciding between the two approaches is a challenging task. In this paper, we aim to facilitate the developers in choosing the right approach for developing their applications. We investigate the advantages and disadvantages of each approach in various scenarios and considers the parameters from different use cases. The main objectives of this research are:

- To assess and compare the two major mobile application development approaches, i.e., native and cross-platform development
- Checking performance, usability and support
- To implement an app using both native and cross-platform and then use various benchmarks to assess certain KPIs
- Recommend which approach to choose in general and depending on the tools and environment available.

This paper focuses on the two major mobile operating systems, Android and iOS. While some cross-platform solutions target more platforms, these two are the most prevalent. We conduct an analysis on native and cross platform, to evaluate certain parameters of interest. Mainly performance, and usability. Most of the focus will be on Flutter, from the cross-platform side and Android, from the native side.

Choosing the development approach or tool for developing Mobile applications has great significance in all aspects of the project, since the investment in a tool that ends up not meeting the desired outcome will waste plenty of resources. This research evaluates the two general approaches of app development, native and cross-platform, and the results can be used as guidelines when making the decision.

The rest of the manuscript is structured as follows: Section 2 describes the background of each development approach. In Section 3, a detailed comparison of the existing studies with our work is presented. Related work is explained in detail. Section 4 provides the research methodology adopted for conducting this study. Results are discussions are provided in Section 5, while Section 6 discusses the best approach based on our proposed evaluation criteria. Finally, the conclusion and future work is provided in Section 7.

1.1 BACKGROUND

1.1.1 Native Application Development

MA Native application is an application developed and targeted at a certain platform. Native apps directly use the platform's main language, tools, framework and APIs, to access and use the resources available for the system, in order to extend the device functionality and offer value. The tools used for developing native apps are often in abundance and have direct support from the platform they are targeting, since it's in the platform's best interest to make the developer's job as pain-free as possible. Tools here mean programming languages, IDEs (Integrated Development Environment), frameworks, libraries, documentation, courses, etc.

Native development also often has a big community, with open-source libraries and frameworks that lets the developer focus on their App's business logic rather than spend time doing something that has already been implemented and tested thoroughly. Given the scope of this paper, the platform means either Android or iOS.

1.1.2 Android Native Development

Android is a Linux-based operating system, meaning its kernel is written with C/C++. The framework layer is written in Java, as well as the APIs. Java is a high-level programming language that is platform-independent. To achieve this, Java runs on top of what is called JVM (Java Virtual Machine). Java code is first compiled into ".class" files that contain byte code, the JVM then interprets the byte code in machine code that depends on the host platform the JVM is run on [2]. Even though Android uses Java, there are some differences. Given the resource limitations mobile devices generally have, some changes had to be made to the java environment in order to better suit mobile needs. Android has two first class supported languages, Java and Kotlin. Given the popularity and robustness of Java, Google made a choice to use it for the Android platform, and for a while, it was the only one with first-class support. However, in Google I/O 2017, the Android team announced first-class support for Kotlin [3]. Kotlin is a concise modern programming language that can run on the JVM, it also can be compiled to JavaScript code or LLVM native code.

1.1.3 iOS native development

Apple launched the App Store in 2008 with 522 apps. To make these apps, the language of choice for Apple was Objective-C, A C-like programming language with object-oriented features. Apple uses objective see for both of its major operating systems, OSX and iOS, with their APIs also written in it. As Objective-C aged apple decided it needs to find a replacement, so in 2014, during their Worldwide Developers Conference, Apple announced Swift, a new modern programming language for iOS and Mac OS applications. Swift was originally a side project for an apple employee, Chris Lattner. But after a while, it gained interest and attention within the company. Swift is a compiled language that uses the LLVM and Objective-C runtime, meaning it can leverage and interact with existing Objective-C code, which allows Swift to directly interact with the iOS framework APIs [4].

1.1.4 Cross-Platform Application Development

The cross-Platform purpose is to solve the problems caused by the fact that native applications are platform-dependent, by making an abstraction that works in more than one platform. To achieve this, there are different approaches categorized into web, hybrid, interpreted and generated apps [5].

1.1.5 Web Approach

Apps that are browser-based web apps take advantage of technologies like HTML and JavaScript to make platform-independent apps. Web apps depend on the browser they run on, which renders the HTML and interprets the JavaScript. This approach means any platform with a browser can run them. Web apps have limited access to the underlying structure of the platform, since they depend on what the browser exposes as capabilities. They also have to be downloaded each time they are used, since there is no installation process [5]. Figure 1 depicts the interaction of the mobile apps with the browser and backend.

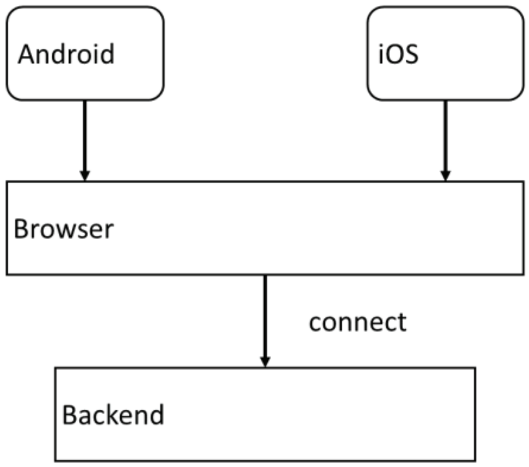

Fig. 1. Web approach

1.1.6 Hybrid Approach

As depicted in Figure 2, Hybrid apps are a mid-ground between Web Apps and Native Apps, they use the native browser, like UIWebView in iOS and WebView in Android, to run web pages. Their difference from Web Apps is that they are packaged and installed on device and have their content saved locally so they don't have to be downloaded each time. They also have access to the underlying capabilities of the platform they work on. An example for them is Cordova [6].

1.1.7 Interpreted Approach

Interpreted apps depend on the underlying tools to interpret the code to platform-specific native code, like some programming platforms as Java does. An example of a software environment that creates

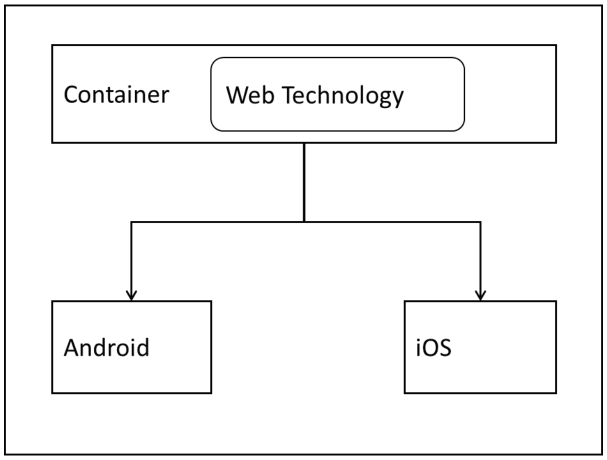

Fig. 2. Hybrid approach

Interpreted apps is Appcelerator Titanium [5].

This approach provides native user interfaces but has downsides like dependence on the tool, such as the case where a new user interface is available in the native platform but is not yet supported by the tool.

1.1.8 Generated Approach

Generated apps are compiled to platform-specific code depending on the target platform, so each platform will have different executable code. An example of a software environment that creates Interpreted apps is Applause [5].

1.1.9 MDA approach

MDA is a design approach that allows development using high-level constructs without having to deal with low-level details. MDA acts as a middleware that abstracts away operating systems, programming languages, etc., allowing focus on the business logic of the product [7]. MDA was defined by the Object Management Group (OMG) [8]. MDA consists of, well, models, as shown in Figure 3.

1.2 Related Work

In the paper "A Comparative Analysis of Cross-platform Development Approaches for Mobile Applications" by Spyros Xanthopoulos from the Aristotle University of Thessaloniki and Stelios Xinogalos from University of Macedonia [5], the authors suggested that the use of native application development technologies imposed "severe constraints", things like multiple development environments and increased maintenance cost were mentioned. The paper evaluates different cross-platform development types, which include, web, hybrid, interpreted and generated apps.

In the white paper "Analysis of Native and Cross-Platform Methods for Mobile Application Development" by Praveen Kumar S [9], the author conducts an analysis on the native and cross-platform by highlighting their respective features, advantages and limitation. The author suggests that in the future the choice of the development approach will become costlier as the process become more complex because of increased mobile device fragmentation.

In the paper "Evaluating Cross-Platform Development Approaches for Mobile Applications" by Henning Heitkötter, Sebastian Hanschke, and Tim A. Majchrzak [10], the authors evaluate cross-platform solutions for mobile, including the most prevalent at the time of the publishing, like PhoneGap and Titanium Mobile. Some of the criteria they used for the evaluation are Look and feel, ease of development Maintainability, scalability and application speed. The authors argue that cross-platform is mature enough that the native approach is not always needed.

In the paper "Cross-platform approach for mobile application development: a survey" by

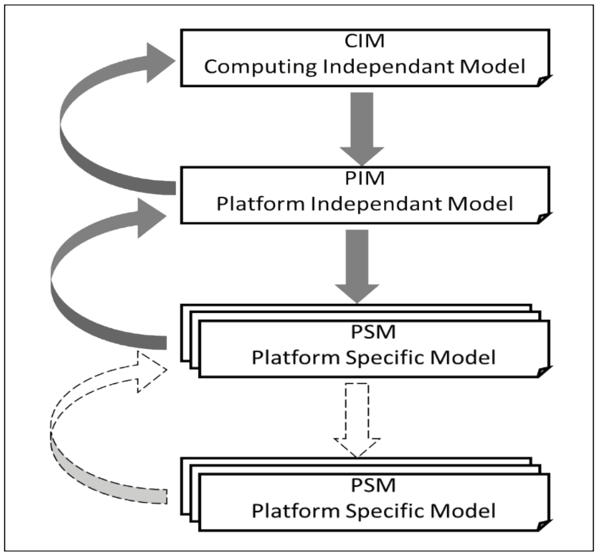

Fig. 3. Model Driven Architecture. Source (Sommer and Krusche, 2013)

LATIF, LAKHRISSI, NFAOUI and ES-SBAI [11], the authors conduct a survey on current cross-platform approaches while it puts emphasis on the MDA (Model Driven Architecture) approach, it also looks into We, Hybrid, Interpreted and cross-compiled approaches. The paper identifies desirable properties of any cross-platform solution, which includes Application Scalability and maintainability, Access to devices features, Security and Development Environment. The paper concludes that a cross-platform solution is favourable to native when time and cost constraints are present, and it recommends a solution using the MDA approach.

In the paper “Survey, Comparison and Evaluation of Cross-Platform Mobile Application Development Tools” [12], The authors discuss the decision criteria for choosing a suitable cross-platform tool. The authors first identify the desired requirements to be met in a cross-platform framework, then they discuss the general architecture of cross-platform development, and finally, they conclude with a survey of several cross-platform solutions (PhoneGap, Titanium, Sencha Touch). The paper concludes that the user experience in cross-platform applications is not as good as with native, but it still offers more potential to reach more users straightforward.

In the paper “Cross-platform mobile development approaches” [13], the authors present a comparison between several cross-platform approaches, including: Runtime, Sources Code Translators. Web-to-native wrapper, App factories and JavaScript frameworks. Some notable criteria present are: The type of the resulting App (Native, hybrid or web), the app size, performance hit (CPU or memory), supported platforms and access to underlying platform APIs. In conclusion the paper emphasizes the need to analyze the desired objective in order to choose a suitable cross-platform tool, and the paper present three factors that help make that choice, which are, programming habits, the importance of native look and feel, and the target OS.

In the paper “Baseline Requirements for Comparative Research on Cross-Platform Mobile Development: A Literature Survey” [14], the authors state how the technical implementations are used to test hypotheses in the computing field, and that research in the mobile field lacked a common baseline. The authors propose a baseline to be used for cross-platform mobile app development research. Their results include which tool to use for each cross-platform approach (like Xamarin. For cross-compiled / Generated), which devices to test on for each major mobile platform and the features to assess. The authors conclude that a signal baseline is not feasible, so they presented several baselines for different types of studies. They also conclude that the approaches and tools change and depreciate over time.

In the paper “Evaluation of cross-platform frameworks for mobile applications” [15], the authors conduct an evaluation of then current cross-platform tool against their native counterpart. The evaluation is done by assigning weights to certain desirable properties like functionality and developer support, and averaging for a final score. The native SDKs got the higher scores overall with the biggest gap appearing in the “Reliability & Performance” category. In conclusion, the authors argue that cross-platform solutions are of value if the performance hit is acceptable for the use case.

There also exist several other studies on native and cross platform development [16-23], however, none of them specifically targeted the comparison of the two approaches as we did in this paper.

2. MATERIALS AND METHODS

The procedure followed in this research include Goal Question Metric, criteria evaluation through investigation and Case Study to further measure different aspects of the competing approaches evaluation through investigation and Case Study to further measure different aspects of the competing approaches.

2.1 Goal Question Metric (GQM)

Goal Question Metric [24] is a top-down approach that works as a measurement mechanism that breaks down the study or project into Goals that need to be reached, Questions to be answered to reach the Goal and Measurements to evaluate said Questions. This paper uses the GQM method and uses the evaluation and case study to answer the questions. Figure 4 depicts the association of the goal with the corresponding questions and metrics.

2.2 Evaluation

To evaluate the two development approaches for mobile, native and cross-platform, the paper identifies desirable characteristics that will be investigated in each approach. The paper uses some of the criteria identified by H. Heitkötter, S. Hanschke and T. A. Majchrzak [10], to test against selected candidates from each approach. These criteria include:

- License and Costs
- Access to platform-specific features
- Long-term feasibility
- Look and feel
- Application Speed
- Development environment
- Ease of development
- Scalability

2.3 Validation

To keep the validity of the thesis reasonably high, GQM is used to link the goal of the study, to the questions and metrics that help reach the goal, as listed in Table 1. Evaluation through investigation or case study is used where appropriate. While the case study provides actual real-world data, the investigation helps fill the gaps of criteria examination in the case study.

2.4 Case Study

The case study is a mobile app that displays a Timer that shows minutes, seconds and a two-digit fraction of a second. The App is a modified version of Alex Sullivan's timer, that is used for the same performance testing purpose [25].

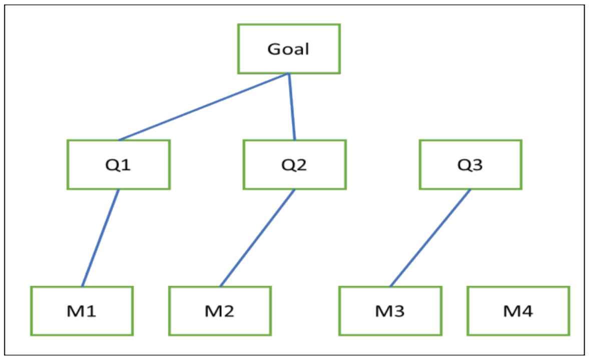

Fig.4. Goal Question Metric Hierarchy

2.5 Instrumentation

Following tools are used in our experiments:

- Android Studio
- Kotlin
- Flutter
- Dart
- Flutter plugins
- Benchmarking tools

2.6 Assumptions and Limitations

The paper won't evaluate all existing cross-platform technologies, and rather it will try to represent major technologies and approaches, choosing one candidate from native (native Android) and one candidate for cross-platform (Flutter). Flutter was chosen because it's a new tool that is gaining traction and because of the lack of studies using it. The performance data will also be affected by choice of and the implementation of the applications, the paper will try to minimize this effect by optimizing the applications to a reasonable degree. Lack of tools and hardware for iOS development limits the evaluation, in the case study, to the results present on android devices.

3. RESULTS AND DISCUSSION

When the App first starts, it saves the initial start time as the difference between the current time and boot time (for Native) or between the current time and time since the Unix epoch, this is called the start

Table 1 Questions and Metrics for Evaluation

Question 1	What approaches are available?
Question 2	What are the characteristic of said approaches?
Question 3	what criteria to assess to choose an approach?
Metric 1	survey available approaches and tools
Metric 2	investigate approaches and choose candidates
Metric 3	evaluate candidates
Metric 4	design set of questions to help choose an approach depending on the use case

time. After that, periodically, the time difference between the current time and start time is calculated and displayed in the timer in the proper format. The rapid, frequent update for the timer is good enough to display performance differences between the two approaches when looking at resource consumption.

The App also showcases the minimum requirements in terms of memory and storage. Since the app functionality is not memory intensive, and neither is it when it comes to storage, then most of the resources are needed for the framework to function. The tests were run on a Samsung Galaxy Note 5 Device, and the profiler bundled with Android studio was used. Figure 5 shows framed screenshots for the App in Android native and Flutter, respectively.

3.1 CPU Readings

CPU readings from the two apps, depicted in Figure 6, demonstrates the overhead that is present when using cross-platform solutions. The native App has a CPU usage of 3% while its flutter counterpart has a 5.5% usage.

3.2 Evaluation

Using the results and observations from the case study, as well as thorough investigation, the native Android and Flutter are evaluated using the criteria mentioned in the previous section.

3.3 License and Cost

Android Open Source Project (AOSP) prefers the usage of Apache 2.0 license [26], which meets their use case for openness and providing more options to manufacturers in means of how to use the platform.

The mentioned license is for Android itself, for development, however, using the Android SDK for native Android development incurs different terms and license [27], which grants a patent to developers to use the SDK, and the sources for the SDK are also available. Flutter is also open-source [28].

3.4 Supported Platforms

Native development, for Android or iOS, involves using the platforms SDK for developing applications targeted only to one platform. Hence native approaches, in the sense of native vs cross-platform, support only one platform.

Flutter builds cross-platform applications that target both Android and iOS [29], which is the main difference between native and cross-platform. The ability to target multiple platforms, with even the infrastructure to even support future platforms, is a very valuable trait.

3.5 Access to platform Specific Features

Native development for Android or iOS gives direct access to all available features of the platform. Flutter, out of the box, gives access to a part of the underlying platform's features in the form of read-made packages, such as, access to biometric

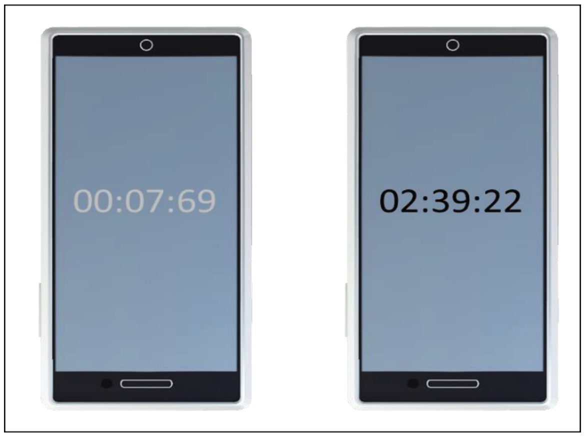

Fig. 5. Timer app in native android (left) &

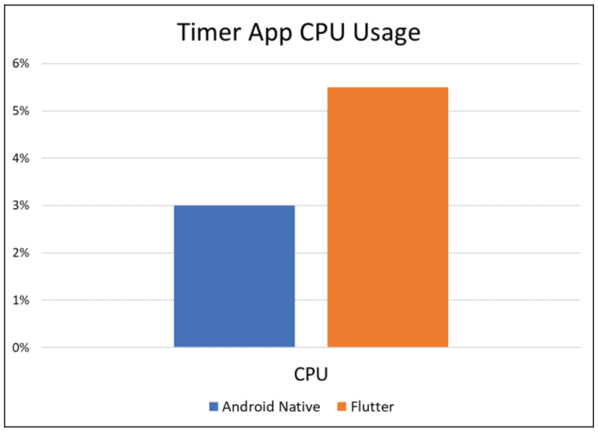

Fig. 6. Cpu comparison

technologies like a fingerprint for authentication [30] and storage access for storing simple key-value pairs [31], but it does not have cross-platform packages for all platform-specific features and APIs, which is a decision explained to made to avoid the issue of "lowest common denominator" [29], which restricts the features of the cross-platform packages to the capabilities available in all platforms (Android and iOS).

For features not available as ready packages, Flutter allows developers to access platform features via message passing, allowing Flutter to message a part of the program that is written in the corresponding native language (Java or Kotlin for Android, Objective C or Swift for iOS), and get a response when the request is handled [32].

3.6 Long Term Feasibility

The native approach, for Android or iOS, is the official approach endorsed by the respective platform, and it's reasonable to assume that it will be supported as long as the platform is still relevant. Even still, some changes can happen in the ecosystem, such as the introduction of Kotlin development for Android [2], and Swift for iOS [33], but the underlying SDKs are still unchanged, and original programming languages can still be used. It can also be seen that each platform is striving to make native development a better experience for developers, which in this case is done by supporting new modern programming languages.

Flutter is a Google product, one of the largest tech giants with around 110 billion dollar earnings in the third quarter of 2018 [34], and also the company behind Android, which is a good indicator for flutters survivability, but it's likely will be ultimately decided with the degree of its success and adaptability, which is too early to tell given how it only recently hit its first stable release [35].

In Conclusion, the native approach is always better supported given that its critical for the platform in question, while the cross-platform approach is affected by the backing of its creators of or the community in case of open-source.

3.7 Look and Feel

Look and feel is platform specific, since users expect applications in a certain platform to have certain look and certain behavior, which is defined by how the native applications look and feel. For Android google use material design [36], while iOS use Human Interface Guidelines [37].

Flutter has widgets, layouts and themes that use Material design for Android and used Cupertino (iOS -like style) for iOS [29], which solves the issue of difference in the look and feel between the different platforms.

The conclusion is that Native has the most authentic look and feel to the platform, but cross-platform solutions like Flutter take a good approach to achieve the desired look and feel across platforms. Also, it can be seen that some applications take the approach of making their applications look and feel the same across platforms, with the intent of making the experience consistent and as a way to focus on the desired experience that the brand needs to convey to customers. In the Later approach, the cross platform can be more appealing in that regard.

3.8 Distribution

Whether it is native or cross-platform, applications can be distributed via the specific platforms app store, as long as they comply with the respective rules and policies.

3.9 Development Environment

Android native development is mostly done using Android studio, the IntelliJ based IDE (Integrated Development Environment) that is reached in features tools. Some of its offerings include things like Instant run, translation editor and APK analyzer [38].

Flutter can be integrated into an android studio; it also has a very straightforward way for SDK installation. Both contribute to a good experience, especially for developers making a move from the Native Android ecosystem.

In conclusion, both approaches give a good development environment, but with the native having generally more tooling available, by the fact it had quite a while to mature.

3.10 GUI Design

Native Android development using Android studio can make use of the build in WYSIWYG tool that facilitates building graphic user interfaces and see the result without the need for time consuming way building and running the application every time a change is done, this is achieved by interpreting the XML files which are used to describe the UI [38].

Flutter does not have a WYSIWYG, but it achieves the desired results in different ways. Flutter uses Dart programming language for both its logic and GUI portions and its uses hot reloading as a means to almost immediately reflect code changes into the running App, and it takes significantly less time to achieve so.

In conclusion, Flutter's hot reload is a very good quality of life feature that does not have a good enough counterpart in the Native approach, despite attempts to give a similar outcome using instant run.

3.11 Maintainability

Although the technologies used in the native affect its maintainability, the fact remains that a native application for both Android and iOS means that two separate code bases negatively affect maintainability. On the other hand, in Flutter and other cross-platform approaches, only one code base exists, which helps maintainability.

3.12 Speed and Cost of Development

Native development, for Android or iOS, requires specific knowledge about the framework, programming language and tools, resulting in doubling the work, increasing either or both of the cost and speed of development.

Flutter is a cross-platform framework, and one of the main points of cross-platform is to run the same code base on multiple platforms. But in reality, other factors can have an effect, and the lowest common denominator problem issue is solved in Flutter by allowing certain functionalities to be implemented separately for each platform. This means two separate implementations, which can become counter-intuitive depending on the type of the application. But nevertheless, the general rule is cross-platform is cheaper and faster. This is supported by the observation that the case study in the previous section took roughly the same time to develop in native and Flutter, while of course, flutter runs on two platforms.

4. BEST APPROACH

Using the outcomes of the evaluation, the following criteria can help decide which approach to choose for development:

4.1 Budget

Budget is the main form factor, the time and resources required vary depending on the nature of the application and the approach used for development. Native apps usually require a developer/team for each OS, and even if it's a single developer/team, there are still two separate code bases, on the other hand Cross-platform apps need one developer/team to get the job done which makes it cheaper. If the budget is not a problem, however, native apps offer uncompromised performance and user experience, making it more favourable in this situation.

4.2 Type of App

The type of the App is a significant factor, typical CRUD apps can be relatively straightforward to make using both approaches, but another kind of apps that require access hardware sensors or device, like a camera app, can become a nuisance to make using a cross-platform approach because the lowest common denominator problem. Native apps, however have direct access to all the APIs that expose device resources, making the approach more ideal for this type of App.

4.3 Developer Background

The developer's previous knowledge can determine how difficult to learn a new development framework, for example, web developers can find themselves at home using cross-platform frameworks that use web technologies or programming languages like react-native.

5. CONCLUSIONS

In this paper, we investigated the two widely used approaches (Native and cross-platform) by developing and evaluating a mobile app. We discussed our results from various perspectives highlighting the advantages and shortcoming of each approach, with native having the upper hand in criteria such as performance and access to platform specific features, and cross-platform showing an advantage in terms of cost and maintainability. We have used the well-known Goal Question Metric (GQM) as a measurement mechanism to breakdown our study into Goals and to answer the Questions for reaching our goals. The cross-platform approach has many established frameworks with different ideas to deliver on the write once promise and Flutter is a promising framework that builds upon the experience gained from the previous frameworks.

The decision of which development approach to use is a costly one, but the answer is not straightforward, it should be decided by the nature of the project, the team developing and the budget. However, based on our findings, native is still the safest approach for mobile application development. In future, the following options can be investigated:

- Address the limitations present in this paper, for example, by evaluation multiple cross-platform frameworks.
- Test different upcoming approaches e.g., Kotlin-native, that promises the best of both worlds, native and cross-platform.
- What if the number of dominant mobile operating systems increase? Is there a point at which native becomes impractical?
- With mobile devices becoming increasingly more performant in terms of hardware, will the performance advantage of the native approach become unnoticeable? Or will the applications become ever more demanding of resources?

6. REFERENCES

1. "Mobile OS market share 2018 | Statista," 2018. [Online]. Available: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/.
2. J. Gosling, B. Joy, G. Steele, G. Bracha and A. Buckley, "The Java® Language Specification, 2015," Java SE, vol. 8, 2016.
3. M. Shafirov, "Kotlin on android. now official," 2017. [Online]. Available: https://blog.jetbrains.com/kotlin/2017/05/kotlin-on-android-now-official/.
4. G. Wells, The Future of iOS Development : Evaluating the Swift Programming Language The Future of iOS Development : Evaluating the Swift Programming, 2015.
5. S. Xanthopoulos and S. Xinogalos, "A comparative analysis of cross-platform development approaches for mobile applications," in Proceedings of the 6th Balkan Conference in Informatics, 2013.
6. Ziflaj, "Native vs Hybrid App Development," 2014. [Online]. Available: http://www.sitepoint.com/native-vs-hybrid-app-development/.
7. R. Soley and the OMG Staff Strategy Group, "Model driven architecture," OMG white paper, vol. 308, p. 5, 2000.
8. S. Roubi, M. Erramdani and S. Mbarki, "A Model Driven Approach for generating Graphical User Interface for MVC Rich Internet Application.," Computer and Information Science, vol. 9, p. 91–98, 2016.
9. P. Kumar, "Analysis of Native and Cross-Platform Methods for Mobile Application Development," 2014.
10. H. Heitkötter, S. Hanschke and T. A. Majchrzak, "Evaluating cross-platform development approaches for mobile applications," in International Conference on Web Information Systems and Technologies, 2012.
11. M. Latif, Y. Lakhrissi, N. Es-Sbai and others, "Cross platform approach for mobile application development: A survey," in 2016 International Conference on Information Technology for Organizations Development (IT4OD), 2016.
12. S. K. Dalmasso, C. Datta, N. Bonnet, and Nikaein, "Survey, comparison and evaluation of cross platform mobile application development tools," in 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), 2013.
13. S. Charkaoui, Z. Adraoui and E. H. Benlahmar, "Cross-platform mobile development approaches," in 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), 2014.
14. A. Biørn-Hansen, T.M. Grønli and G. Ghinea, "Baseline Requirements for Comparative Research on Cross-Platform Mobile Development: A Literature Survey," in Norsk Informatikkonferanse

- 2017, 2017.

15. Sommer and S. Krusche, "Evaluation of cross-platform frameworks for mobile applications," Software Engineering 2013-Workshopband, vol. 215, no. January, pp. 363-376, 2013.
16. P. Nawrocki, K. Wrona, M. Marczak, and B. Sniezynski. A Comparison of Native and Cross-Platform Frameworks for Mobile Applications. Computer, 54(3), 18-27 (2021)
17. D. Inupakutika, S. Kaghyan, D. Akopian, P. Chalela, and A.G. Ramirez. Facilitating the development of cross-platform mHealth applications for chronic supportive care and a case study. Journal of biomedical informatics, 105, p.103420 (2020).
18. A. Biørn-Hansen, C. Rieger, T. M. Grønli, T. A. Majchrzak, and G. Ghinea, An empirical investigation of performance overhead in cross-platform mobile development frameworks. Empirical Software Engineering, 25, pp.2997-3040 (2020)
19. M. Isitan, and M. Koklu. "Comparison and Evaluation of Cross Platform Mobile Application Development Tools." International Journal of Applied Mathematics Electronics and Computers 8, no. 4: 273-281 (2020).
20. A. Biørn-Hansen, T.M. Grønli, and G. Ghinea. Animations in cross-platform mobile applications: An evaluation of tools, metrics and performance. Sensors, 19(9), p.2081 (2019).
21. M. Martinez, "Two datasets of questions and answers for studying the development of cross-platform mobile applications using Xamarin framework." In IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft), pp. 162-173. IEEE, 2019.
22. K. Vassallo, G. Lalit, P. Vijay, and K. Ramesh. "Contemporary technologies and methods for cross-platform application development." Journal of Computational and Theoretical Nanoscience 16, no. 9, 3854-3859 (2019).
23. I. Swarna, P. James, and A. Randy. "Cross-Platform Analysis and Development of Online Catering Platform (Kunyahku)." Journal of Applied Information, Communication and Technology 7, no. 2, 79-89 (2020).
24. V. R. Basili, G. Caldiera and H. D. Rombach, "The goal question metric approach," Encyclopedia of Software Engineering, p. 528–532, 1994.
25. Sullivan, Examining performance differences between Native, Flutter, and React Native mobile development, 2018.
26. "Content License | Android Open Source Project," 2018. [Online]. Available: https://source.android.com/setup/start/licenses.
27. "Terms and conditions | Android Developers," [Online]. Available: https://developer.android.com/studio/terms.
28. "Flutter License," 2018. [Online]. Available: https://github.com/flutter/flutter/blob/master/LICENSE.
29. "FAQ - Flutter," 2018. [Online]. Available: https://flutter.io/docs/resources/faq#what-devices-and-os-versions-does-flutter-run-on.
30. "local_auth | Flutter Package," 2018. [Online]. Available: https://pub.dartlang.org/packages/local_auth.
31. flutter.dev, "shared_preferences | Flutter Package," 2018. [Online]. Available: https://pub.dartlang.org/packages/shared_preferences.
32. "Writing custom platform-specific code - Flutter," 2017. [Online]. Available: https://flutter.io/docs/development/platform-integration/platform-channels.
33. "Swift - Apple Developer," 2018. [Online]. Available: https://developer.apple.com/swift/.
34. "Alphabet Inc (GOOG) 2018 3Q Earnings," 2018. [Online]. Available: https://www.sec.gov/Archives/edgar/data/1652044/000165204418000035/goog10-qq32018.htm.
35. "Flutter 1.0," 2018. [Online]. Available: https://developers.googleblog.com/2018/12/flutter-10-googles-portable-ui-toolkit.html.
36. "Design for Android," 2018. [Online]. Available: https://developer.android.com/design.
37. "iOS Human Interface Guidelines," 2018. [Online]. Available: https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/.
38. "Android Studio features," 2018. [Online]. Available: https://developer.android.com/studio/features.