iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 12 **Issue:** IV **Month of publication:** April 2024

DOI: https://doi.org/10.22214/ijraset.2024.60066

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 12 Issue IV Apr 2024- Available at www.ijraset.com

Evaluation of Cross-Platform Technology Flutter Using Different System Environments

Prof. S. P. Akarte[1], Khushi Kedia[2], Arpan Ade[3], Anuj Yewale[4], Kanhaiya Lasurkar[5]
Department of Computer Science and Engineering, Prof. Ram Meghe Institute of Research & Technology, Amravati, Maharashtra, India

Abstract: ***The aim of the following thesis was to evaluate the cross-platform technology Flutter based on a user perspective. The key aspects investigated were user-perceived performance such as startup time and application size. Additionally, the user-perception was also a key feature investigated. To evaluate the cross-platform technology Flutter, one social media applications was developed.***
For evaluating the developer perception between Flutter and native applications, a user study was conducted. The results suggested that if the performance of the application is vital for the users, a Flutter application is most likely not suitable. The user perception study showed that there were no significant differences between the developed applications. No significant differences between the Flutter applications and the native application for either the Android or the iOS platform. Thus, a Flutter application can be a suitable alternative to a native Android application from the user's perspective
Keywords: Flutter Framework, Android application, iOS application, App performance, Real-time working

I. INTRODUCTION

In the last decade of the smartphone era, there have mainly been two platforms dominating the market: Android and iOS [1, 2, 3, 4]. It has resulted in developers building applications with identical functionality for separate platforms with different codebases [5, 6, 7, 8]. There are several cross-platform technologies available aiming to solve this issue, giving developers the opportunity to "write once, run anywhere". One example of such a platform is Flutter which is created by Google. Flutter aims to solve the challenge of developing applications for different platforms with a single codebase. Thus, achieving write once, run on several platforms, such as Android and iOS. [9, 10].

The following thesis evaluates the cross-platform technology Flutter based on a user perspective. The key aspects investigated are user-perceived performance such as startup time and application size. The gist of this thesis evaluation is based on developing social media application. One native application each for Android and iOS, and two Flutter applications for Android and iOS using the same codebase. This will result in three codebases and four sample applications. Thereafter, a user perception study will be conducted on the social media application. Additionally, the performance will be measured while the participants in the study use the application.

II. LITERATURE REVIEW

Attempting to evaluate a whole cross-platform technology is a substantial effort. Considering that Flutter supports multiple platforms such as Android, iOS, macOS, and web. Due to limited time and scope, this thesis will solely focus on evaluating Flutter from a user-centered perspective for the Android and iOS platforms.

Google is not the first company attempting to create a cross-platform solution, and certainly not the last. There have been several cross-platform solutions earlier, and multiple of those solutions have targeted the mobile platforms Android and iOS. Therefore, a fair amount of research has been conducted on cross-platform solutions for Android and iOS. There was one study by Hansson and Vidhall [11] that significantly influenced this thesis. They evaluated the mobile cross-platform technology named React Native for both the Android and iOS platforms. Similar to this thesis, they developed sample applications both natively and using the cross-platform technology under evaluation. Those sample applications were the fundamentals of their research. The applications were used to test application performance and the user's perspective of the different applications. Hansson and Vidhall [11] study did not emphasize the user's perspective, neither was it their main focus. They evaluated the cross-platform technology from a broader perspective.

However, one of their research methods wasdeemed to be applicable for this thesis. In particular, they used a user experience questionnaire for understanding the user's perspective. The same method was used for this thesis. All of the previous work has conducted their research focusing on one or a few aspects. Aspect such as application size, development productivity, look and feel experience, performance, taxonomy, UI/UX, etc. Cross-platform solutions are broad and consist of multiple technologies. It is hard or near impossible to research the whole technology and conclude something objectively. Studies regarding mobile application taxonomy tend to split applications into two groups. Applications that run on one platform and cross-platform applications which run on multiple platforms. For single platform applications, most of the studies seems to agree on naming native application [13, 14], except one study by El-Kassas et al. [12]. All of the studies define cross-platform applications in distinctive ways. This thesis will not address the differences in mobile application taxonomy. The thesis will use the naming convention of native application, for applications that run on a single platform. The naming cross-platform application will be used for for applications that run on multiple platforms. This thesis will not address nor investigate the differences in cross-platform taxonomy in regards to the UI toolkit Flutter. There are a plethora of studies related to cross-platform applications. It would be challenging to summarize every aspect of research in the field of cross platform applications. For the purpose of this thesis, thorough research and investigation have been conducted in previous work related to this thesis and its aim. The following sections, sample applications, look and feel, and performance is highly relevant to this thesis.

III.MODEL VIEW

According to Sorensen and Mikailesc [15], developers have been using different design patterns for building applications since the day it was necessary to have user interfaces. Some of these examples are the Model-View Controller(MVC) and the Model-View-Presenter(MVP) pattern. What is presented to the user is the View, necessary data that is visible in the view is the Model, and the Controller or the Presenter ties the two together. In 2005, John Gossman presented another design pattern named Model-View-ViewModel (MVVM). Similar to a Controller or Presenter, the ViewModel glues the Model and ViewModel together. However, in contrast to a Controller or a Presenter, the ViewModel does not hold any references to the view. The ViewModel exposes data models and objects contained in the view. The responsibilities of the ViewModel and View are now different than in the previous design patterns. The design pattern is often visualized linearly (see above fig.). [15, 16]

Fig. 1 The Model-View-View-Model (MVVM) design pattern.

The MVVM pattern is often visualized linearly to point out the flow of data and information. The model is responsible for accessing different data sources (e.g., databases, files, or servers). In general, the model often tends to be thin in the MVVM implementation. The View represents the appropriate format whether it is graphical or non-graphical, reflecting the state of the data. It collects user interactions and events. Similar to the Model, Views in MVVM are thin and contains minimal code. It only contains code that is required to make the View work and allow user interactions.

In MVVM, most of the code is in the ViewModel. The ViewModel should represent the view state and is expected to behave according to the view logic, i.e. the user interactions. ViewModel handles the communication between the Model and the View. It passes all the necessary data between the Model in View in such forms that they can digest the data. Necessary validation is performed in the ViewModel. [31] Kouraklis [31] describes this pattern as the components work in sets of two. The View is aware of the ViewModel, updates the ViewModel's properties, and tracks any changes that occur in the ViewModel. As seen in fig. 2.2, the ViewModel does not hold any references to the View. Similarly, the Model is not aware of the ViewModel or the View. Only the ViewModel has a reference to the Model. The ViewModel passes events and data to the Model, as they are pushed by the view in forms that the Model can interpret. As the ViewModel holds a reference to the Model, it tracks any changes in the Model and consequently pushes necessary signals to the View. The three mentioned design patterns MVC, MVP, and MVVM are common design patterns for both Android and iOS development [17, 18]. In research by Lou [17], he concludes that implementing MVP and MVVM patterns results in a lower coupling level (i.e. superior testability and modifiability) and consumed less memory than using the MVC pattern. There was no significant performance difference between the MVP and MVVM patterns.

The unequal variance t-test is an adaption of the Student's t-test. The Student's t-test assumes an equal variance were the first test method does not. As the name implies, conducting a t-test with unequal variances is based on the assumption that the variance is unequal and/or has an unequal sample size. Which results in the t-test using unequal variances produce more reliable results than the Student's t-test. In practice, it is more like the sample size is unequal. Therefore, Ruxton [19] suggests that using a t-test with unequal variance should always be used in preference to the Student's t-test or Mann–Whitney U test. The t-test with unequal variance is formulated as: t = X1 – X2 r s 2 1 N1 + s 2 2 N2 (2.1) where Xj, sj and Nj are the jth sample mean, sample standard deviation, and sample size, respectively j ∈ {1, 2}. If the t-value is below 0.05 there is a significant difference between the group with a 95% confidence.

IV. PROPOSED METHODOLOGY

Flutter is Google's approach to cross-platform application development. Google themselves describes Flutter as a UI toolkit. With Flutter, developers can build applications for mobile, web, and desktop from a single codebase 2. Web support is currently in a beta phase 3, while desktop (Linux, macOS, Windows) support is still in the alpha phase 4. Declarative UI Flutter is a declarative framework, meaning Flutter builds the UI to reflect the current state of the application 5. In comparison to imperative programming, when an application state changes, the UI also needs to be handled and updated to reflect the latest application state. In declarative programming the application changes when the UI rebuilds to reflect the current application state. One of its benefits is that there is only one code path for any state of the UI. The UI is described once for any state. A potential drawback might be that it rebuild itself as soon as the application state changes, as it might be performance heavy. Architectural layers the design of Flutter is based on layering the system. Each layer contains independent libraries that depend on the layer below. No independent library has prioritized access to the layer below. Every part of the framework level is designed to be optional and replaceable.

To evaluate the cross-platform technology Flutter, multiple sample applications were developed. Firstly, two identical sample applications were developed by using the UI-toolkit Flutter with cross-platform support for Android and iOS. Afterward, the same sample application was developed natively for both Android and iOS. This resulted in developing the same application three times with three different code bases. The native Android application was compared to the Flutter developed Android application, and the native iOS application was compared to the Flutter developed iOS application. When choosing the features of the sample applications, two major aspects were considered: complexity and UI/UX. Both of these aspects were chosen to develop a sample application that would answer the research questions.

In the complexity varied as there were different functionalities implemented. Both Abrahamsson and Berntsen [19] and Fredrikson [18] developed applications that did not use any external APIs an therefore no internet connection was needed. Both of their sample applications were dependent on using mock data.

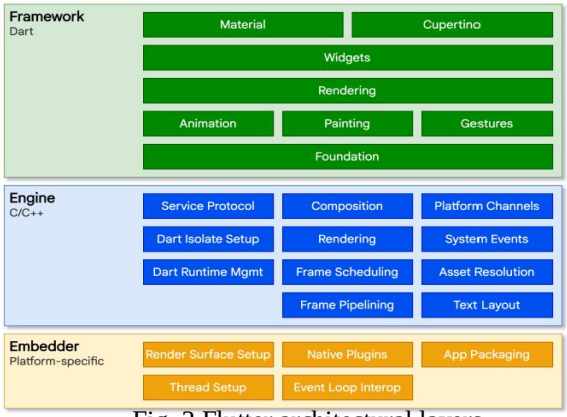


Fig. 2 Flutter architectural layers.

Developers will interact with Flutter using the Flutter framework. It provides a reactive framework with a set of platforms, layout, and foundation widgets. The framework is written in Dart language. It is also the language used when developing Flutter applications.9 The core of Flutter is the Flutter engine, which is developed using C++. The engine exposes itself to the Flutter framework through the dart library dart:ui. It wraps the engine's C++ code in Dart classes.10 The engine implements Flutter's core libraries, including animation and graph ics, file and network I/O, accessibility support, plugin architecture, and a Dart runtime and compiles toolchain. It is responsible for rasterizing visuals when a new frame needs to be painted.

Social media applications developed by Evert [13] and Hansson and Vidhall [11] needed the internet as they communicated with different APIs. Evert [13] application used an external REST-API for searching users on GitHub and view their public repositories using two different endpoints. Hansson and Vidhall [11] developed a home automation application that also used a REST-API, although using a private internal API. Both Evert [13] and Hansson and Vidhall [11] intentionally developed sample applications that would be sufficiently complex applications with characteristics of real-world applications.

Similar to previous studies, the aim when developing the sample applications for this research, was that the applications would be sufficiently complex enough with characteristics of real-world applications. In relation to developed applications for previous studies, the sample applications for this research also needed to at least have equivalent complexity.

To achieve this, it was deemed that the developed sample applications needed to use real data and a REST-API with at least three different endpoints. This would ensure that the sample applications would be more complex than Evert [13] and Fredrikson [14] sample applications.

As mentioned, authors in previous studies developed sample applications that had limited functionalities. However, the applications had at least two views. Therefore, the developed sample applications for this research at least needed three views. It would ensure that the sample applications were not too limited.

With these conditions, it was assumed sufficient enough to have at least the same complexity as previously developed sample applications. It was also assumed to mimic a production application so it could be used to make valid conclusions.

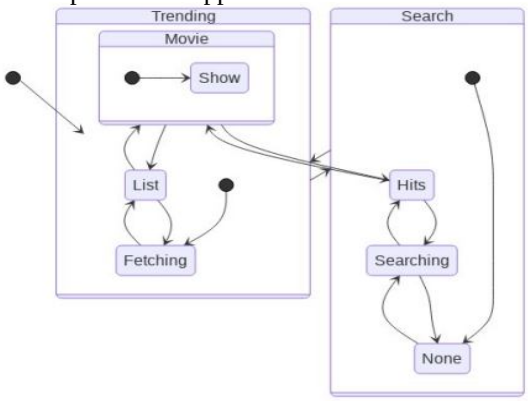


Fig. 3 State diagram of the developed sample applications

The state diagram reflects the user flow and different states for all of the developed sample applications. As they are identical for each developed sample application. This means, when the user starts the app, the trending view is shown immediately and it fetches trending movies from the TMDB API. After startup, the user can change between being in the trending view or search view anytime. Thus, the outer state is always in the Trending or Search state. When the user selects a movie from either the trending list or on a hit from the search results, the inner state is changed to the Movie state. The user can return to the previous state from the Movie state.

V. RESULT ANALYSIS

A. Size

On both Android and iOS, the installation package size and the installed appli cation size were larger than the native applications package size and installed application size. This is due to the overhead of the Flutter engine and frame work. Even if an application is quite minimal, such as the sample applications, the Flutter core is still necessary. Hence, the difference in size between Flut ter applications and native applications will not increase linearly, even if more functionality is added to the applications. The Android installer .aab for the Flutter application was 16.8 MB larger than the native application. This corresponds to the Flutter .aab being approxi mately 4 times larger than the native application .aab. The iOS installer .ipa for the Flutter application was 72.2 MB larger than the native application. This corresponds to the Flutter .ipa being approximately 20 times larger than the native application .ipa.

	.aab	.ipa
Native	5.4 MB	3.9 MB
Flutter	22.5 MB	76.1 MB

Fig. 4 Application package sizes of the developed sample applications.

The installed application size differs from the application package size, as it gives different install files for different devices. Meaning, the installed application sizes also differ among different devices. The result of installed applications on two different test devices is shown in table.

	Samsung Galaxy S9 (Android 10)	iPhone XS (iOS 14.3)
Native	5.4 MB	3.8 MB
Flutter	35.73 MB	32.5 MB MB

Fig. 5 Installed application size of the sample application on test devices.

On Android, the installed Flutter application was 30.33 MB larger than the native application on a Samsung Galaxy S9 using Android 10. On iOS, the installed Flutter application was 28.7 MB larger than the native application on an iPhone XS using iOS 14.3. This corresponds to the Flutter application was approximately seven times larger and nine times larger than the native sample Android application and respectively sample iOS application.

B. Startup time

On both Android and iOS, the Flutter developed applications took a longer time to start than the native applications. For both platforms, different devices from different years were used. Thus, both old and new devices with different hardware were used. Different OS versions were also being used in the devices. Some devices used the latest version, and some devices were not using the latest version. The fastest startup times were measured on the newest devices, while the slowest startup times were measured on the oldest devices. Having the latest OS version available also seemed to result in a better performance.

The native Android application was tested on seven different devices, and the Flutter application was tested on another six android devices. In a total of 13 unique devices, where all of them were different models. The Flutter developed application startup median was approximately 130% longer than the native application startup median. Thus, native Android application results in a better user-perceived performance in regards to startup time.

The slowest startup time was 826 ms on a Samsung Galaxy S8 using Android 9 for the native application and 936 ms on a Samsung Galaxy S9 using Android 10 for the Flutter application. The fastest startup time for the native application was 77 ms on an OnePlus using Android 9 and 128 ms on a Samsung Galaxy S20 5G using Android 10 for the Flutter application.

The devices which resulted in the lowest startup time were also the oldest devices used in the study. Samsung Galaxy S20 5G was the newest device used in the test. The model of the OnePlus which measured 77 ms for startup time could not be identified from the monitoring tool.

	Lower bound	Median	Upper bound
Native Android	77 ms	463 ms	826 ms
Flutter Android	128 ms	606 ms	936 ms

Fig. 6 Startup time of the developed Android sample applications.

The native iOS application was tested on five different devices, and the Flut ter application was tested on another four devices. In a total of nine different devices. However, there were only three different models. The Flutter de veloped application startup median was approximately 230% longer than the native application startup median. Thus, native iOS application results in a better user-perceived performance in regards to startup time.

The slowest startup time was 716 ms on an iPhone XR using iOS 14.2 for the native application and 3.97 s on an iPhone 6 Plus using 12.5 for the Flutter application. The fastest startup time for the native application was 156 ms on an iPhone XS using iOS 14.3 and 105 ms on an iPhone XS using iOS 14.3 for the Flutter application.

The iPhone 6 Plus was the oldest device and had the oldest version among the three devices tested. It resulted in the longest startup time for the Flutter application. The iPhone XS was among the newest devices and use the newest version among the test devices. It resulted in the fastest startup time for both the native application and the Flutter application.

	Lower bound	Median	Upper bound
Native iOS	156 ms	222 ms	716 ms
Flutter iOS	105 ms	522 ms	3.97 s

Fig. 7 Startup time of the developed iOS sample applications.

VI. CONCLUSION

This thesis aimed to answer the question "can a Flutter developed application be a viable alternative for native application from the user's perspective?". Based on the results, a few conclusions can be drawn.

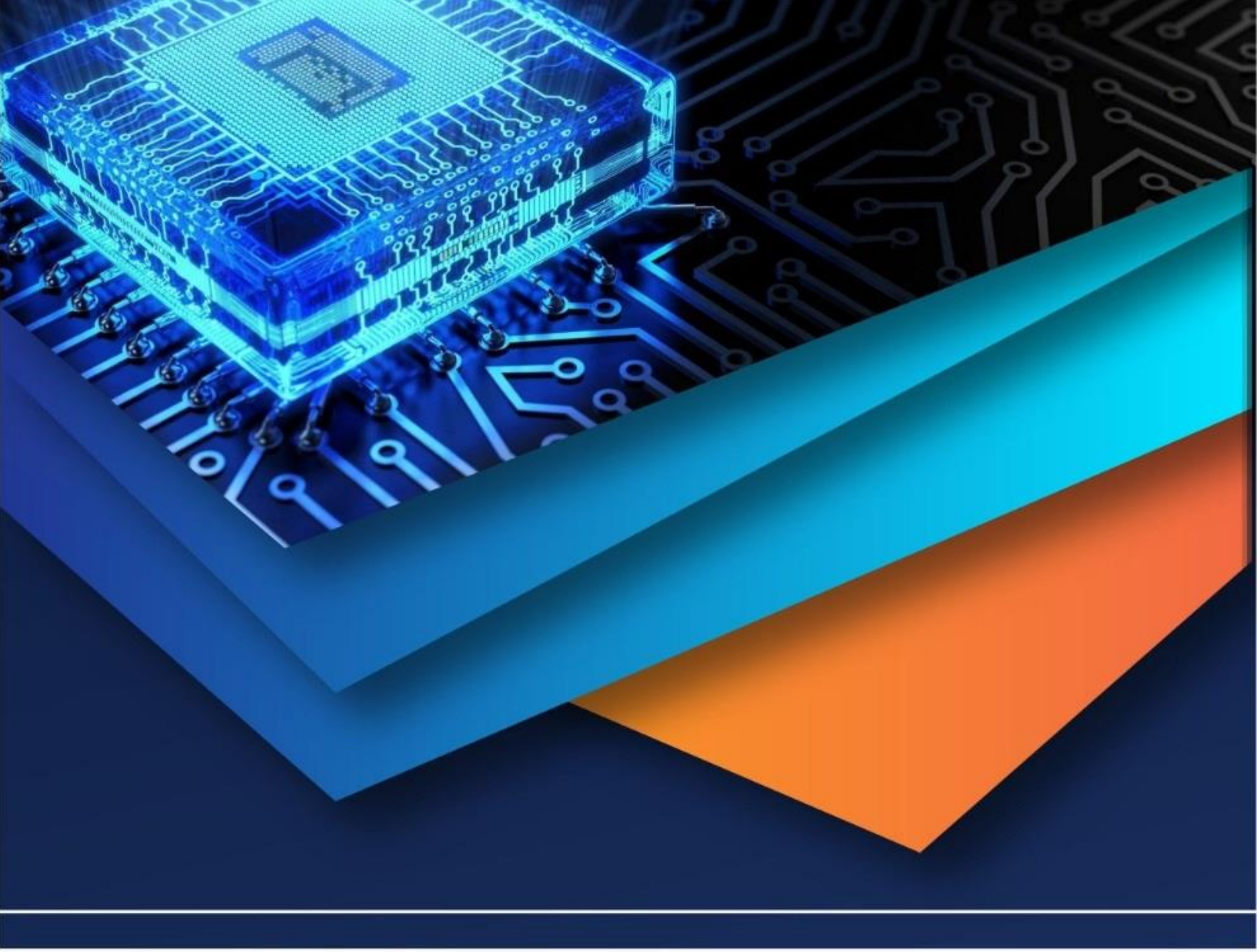
If the application size of the application is vital for the users, a Flutter application is most likely not suitable. However, if it is of less importance, a Flutter application might be a viable alternative to a native application from a user's perspective.

The user-centered perspective was broken down into two smaller focus areas, user-perceived performance and user perception. In both aspects, the native applications seemed to perform better and leave a better impression on the users.

In terms of performance from a user's perspective, Flutter developed applications had a larger application size. The results also indicates the Flutter applications longer time to start. However it is uncertain due to significant variance in measured startup times. One of the main reasons of the inferior user perceived performance may be due to Flutter's internal workings and its built-in overhead.

REFERENCES

[1] Margaret Butler. "Android: Changing the mobile landscape". In: IEEE pervasive Computing 10.1 (2010), pp. 4–7.
[2] Aijaz Ahmad Sheikh et al. "Smartphone: Android Vs IOS". In: The SIJ Transactions on Computer Science Engineering & its Applications (CSEA) 1.4 (2013), pp. 141–148.
[3] Markus Pierer. "Mobile platform support". In: Mobile Device Management. Springer, 2016, pp. 37–42.
[4] Kristiina Rahkema and Dietmar Pfahl. "Empirical study on code smells in iOS applications". In: Proceedings of the IEEE/ACM 7th International Conference on Mobile Software Engineering and Systems. 2020, pp. 61–65.
[5] Iván Tactuk Mercado, Nuthan Munaiah, and Andrew Meneely. "The impact of cross-platform development approaches for mobile applications from the user's perspective". In: Proceedings of the International Workshop on App Market Analytics. 2016, pp. 43–49.
[6] Henning Heitkötter, Sebastian Hanschke, and Tim A Majchrzak. "Evaluating cross-platform development approaches for mobile applications". In: International Conference on Web Information Systems and Tech nologies. Springer. 2012, pp. 120–138.
[7] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. "Comparison of cross-platform mobile development tools". In: 2012 16th International Conference on Intelligence in Next Generation Networks. IEEE. 2012, pp. 179–186.
[8] Spyros Xanthopoulos and Stelios Xinogalos. "A comparative analysis of cross-platform development approaches for mobile applications". In: Proceedings of the 6th Balkan Conference in Informatics. 2013, pp. 213–220.
[9] Frank Zammetti. "Flutter: A Gentle Introduction". In: Practical Flutter: Improve your Mobile Development with Google's Latest Open-Source SDK. Berkeley, CA: Apress, 2019, pp. 1–36. isbn: 978-1-4842-4972-7.
[10] Nikita Kuzmin, Konstantin Ignatiev, and Denis Grafov. "Experience of Developing a Mobile Application Using Flutter". In: Information Sci ence and Applications. Springer, 2020, pp. 571–575
[11] Niclas Hansson and Tomas Vidhall. "Effects on performance and usability for cross-platform application development using react native". MA thesis. Linköpings universitet, 2016 .
[12] Wafaa S. El-Kassas et al. "Taxonomy of Cross-Platform Mobile Applications Development Approaches". In: Ain Shams Engineering Journal 8.2 (2017), pp. 163–190. issn: 2090-4479.
[13] Anna-Karin Evert. "Cross-Platform Smartphone Application Development with Kotlin Multiplatform: Possible Impacts on Development Productivity, Application Size and Startup Time". MA thesis. KTH Royal Institute of Technology, 2019.
[14] Rasmus Fredrikson. "Emulating a Native Mobile Experience with Cross platform Applications". MA thesis. KTH Royal Institute of Technology, 2018.
[15] Erik Sorensen and M Mikailesc. "Model-view-ViewModel (MVVM) design pattern using Windows Presentation Foundation (WPF) technol ogy". In: MegaByte Journal 9.4 (2010), pp. 1
[16] John Kouraklis. "MVVM as Design Pattern". In: MVVM in Delphi: Architecting and Building Model View ViewModel Applications. Berkeley, CA: Apress, 2016, pp. 1–12. isbn: 978-1-4842-2214-0
[17] Tian Lou. "A comparison of android native app architecture–mvc, mvp and mvvm". In: Eindhoven University of Technology (2016).
[18] Mariam Aljamea and Mohammad Alkandari. "MMVMi: A validation model for MVC and MVVM design patterns in iOS applications". In: IAENG Int. J. Comput. Sci 45.3 (2018), pp. 377–389.
[19] Graeme D Ruxton. "The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test". In: Behavioral Ecology 17.4 (2006), pp. 688–690



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY